1302459

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

March 31, 2005

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: ***60/554,051***
FILING DATE: ***March 16, 2004***
RELATED PCT APPLICATION NUMBER: ***PCT/US05/07145***

UNITED STATES PATENT AND TRADEMARK OFFICE
DEPARTMENT OF COMMERCE

Certified by

Jon W. Dudas

Under Secretary of Commerce
for Intellectual Property
and Director of the United States
Patent and Trademark Office

16138 U.S.PTO
031604

13441 U.S.PTO
60/554051
031604

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

Express Mail Label No. ER832351172US

INVENTOR(S)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
THOMAS	ZENG	SAN DIEGO, CALIFORNIA

Additional inventors are being named on the ______ *separately numbered sheets attached hereto*

TITLE OF THE INVENTION (500 characters max)

A playlist play method and Protocol to enable traverse Network Address Translators (NAT) and interact with Firewalls, authenticate firewall transversal protocols, map interactive connectivity establishments and announce signal end of stream

Direct all correspondence to: **CORRESPONDENCE ADDRESS**

[✓] Customer Number: 35114

OR

[ ] Firm or Individual Name	ALCATEL USA				
Address					
Address					
City		State		Zip	
Country		Telephone		Fax	

ENCLOSED APPLICATION PARTS *(check all that apply)*

[✓] Specification *Number of Pages* 112
[ ] CD(s), Number ______
[ ] Drawing(s) *Number of Sheets* ______
[ ] Other (specify) ______
[ ] Application Data Sheet. See 37 CFR 1.76

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT

[ ] Applicant claims small entity status. See 37 CFR 1.27.
[✓] A check or money order is enclosed to cover the filing fees.
[✓] The Director is herby authorized to charge filing fees or credit any overpayment to Deposit Account Number: 02-3979
[ ] Payment by credit card. Form PTO-2038 is attached.

FILING FEE Amount ($): 160.00

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

[✓] No.
[ ] Yes, the name of the U.S. Government agency and the Government contract number are: ______

[Page 1 of 2]

Respectfully submitted,

SIGNATURE [illegible]

Date MARCH 16, 2004

REGISTRATION NO. 48,134
(if appropriate)

TYPED or PRINTED NAME DAVID A. CORDEIRO

Docket Number: 134188PSP1

TELEPHONE (818) 878-5080

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

16138 U.S.PTO
031604

FEE TRANSMITTAL for FY 2004

Effective 10/01/2003. Patent fees are subject to annual revision.

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT	($) 160

Complete if Known	
Application Number	
Filing Date	MARCH 16, 2004
First Named Inventor	THOMAS ZENG
Examiner Name	
Art Unit	
Attorney Docket No.	134188PSP1

METHOD OF PAYMENT *(check all that apply)*

☑ Check ☐ Credit card ☐ Money Order ☐ Other ☐ None

☑ Deposit Account:

Deposit Account Number	02-3979
Deposit Account Name	Michael B. Brooks

The Director is authorized to: *(check all that apply)*

☐ Charge fee(s) indicated below ☑ Credit any overpayments

☑ Charge any additional fee(s) or any underpayment of fee(s)

☐ Charge fee(s) indicated below, except for the filing fee

to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1001	770	2001	385	Utility filing fee	
1002	340	2002	170	Design filing fee	
1003	530	2003	265	Plant filing fee	
1004	770	2004	385	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	160

SUBTOTAL (1) ($) 160

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

			Extra Claims		Fee from below		Fee Paid
Total Claims		-20** =		X		=	
Independent Claims		- 3** =		X		=	
Multiple Dependent						=	

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	86	2201	43	Independent claims in excess of 3
1203	290	2203	145	Multiple dependent claim, if not paid
1204	86	2204	43	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($)

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for *ex parte* reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	420	2252	210	Extension for reply within second month	
1253	950	2253	475	Extension for reply within third month	
1254	1,480	2254	740	Extension for reply within fourth month	
1255	2,010	2255	1,005	Extension for reply within fifth month	
1401	330	2401	165	Notice of Appeal	
1402	330	2402	165	Filing a brief in support of an appeal	
1403	290	2403	145	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,330	2453	665	Petition to revive - unintentional	
1501	1,330	2501	665	Utility issue fee (or reissue)	
1502	480	2502	240	Design issue fee	
1503	640	2503	320	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	
1809	770	2809	385	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	770	2810	385	For each additional invention to be examined (37 CFR 1.129(b))	
1801	770	2801	385	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify) ____________

*Reduced by Basic Filing Fee Paid

SUBTOTAL (3) ($)

SUBMITTED BY

(Complete *(if applicable)*)

Name (Print/Type)	MICHAEL B. BROOKS	*Registration No. (Attorney/Agent)*	39,921	*Telephone*	818-225-2920
Signature	[illegible]			*Date*	MARCH 16, 2004

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Note there is a typo below: it said:
.. whether this has a change to ever become the basis of a playlist
RFC.

I meant:

whether this has a chance to ever become the basis of a playlist RFC.

This Playlist Play method is not in RTSP standard (rfc2326), and could
be
considered an extension to that standard -- in fact, we are considering
submitting a proposal to IETF in the future.

```
>
> Near the end of the playlist SDD review  yesterday, we touched upon
the
> idea of using a new method, in lieu of "SET_PARAMETER", to request a
new
> playlist update or to seek to a new clip within the same playlist.
>
> Although we originally thought that we were too late, schedule-wise,
to
> make such a change, upon researching RTSP RFC more carefully,
thinking
> through it more, and talking to a few people after the meeting, I
have
> realized that, switching to a new method would most likely shorten
the
> overall developement cycle for playlist feature (i.e., 3.4.2
release).
>
> this release will affect the ease of error handling of the protocol
between the
> streamer and the playlist client, the flexiblity of such protocol to
adapt
> to future requirement changes (from Sony or from other customers),
and
> eventualy (perhaps less importantly), whether this has a change to
ever
> become the basis of a playlist RFC.
>
> 1. The two options
>    First, allow me to present the two options in front of us.
> *     SET_PARAMETER
>        This is the approach that we have taken probably since the
> beginning of SDD work. Dave has clearly documented it. Here is the
excerpt
> from page 9 of Dave's latest SDD (version 0.4):
>
>       ...
>       14.     Client application requests a new playlist after a
specified
> amount of time after providing user feedback.
>       15.     PMA server returns a new playlist file and URL.
>       16.     Client application calls a new SET_PARAM API on player
> engine and passes it the playlist URL to request streaming from the
new
> playlist.
>       17.     Player engine sends an RTSP SET_PARAMETER command to the
SM
> passing it the playlist URL.
>       18.     SM returns the NPT (normal play time) value at which the
> switch to streaming from the new playlist will occur.
>       19.     The player engine returns this NPT time value back up to
the
> player application so that it can update the display appropriately.
>
> An example SET_PARAMETER to update playlist and jump directly to clip
8,
> at end of current clip, is shown below:
>
>                       CLIENT REQUEST
>                       SET_PARAMETER rtsp://example.com/public/foo
RTSP/1.0
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 40
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_index:  8
>                       x-pv-activate: End-Of-Clip
>
>                       STREAMING MODULE RESPONSE
>                       RTSP/1.0 200 OK
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 24
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_index:  8
>                       x-pv-clip_ts: 65000
>
>
> *     PLAYLIST_PLAY
>
> In stead of using SET_PARAMETER in step 17 above, we follow the
spirit of
> RTSP extension mechanism (see section 1.5 of RFC2326), and use a new
> method, called PLAYLIST_PLAY, to support the new feature for seeking
> across playlists. Here is how it looks, in lieu of the message
exchange
> above:
>
>                       CLIENT REQUEST
>                       PLAYLIST_PLAY rtsp://example.com/public/foo
RTSP/1.0
>                       CSeq: 7
>                       Session: 1234567890
>                       Range:
```

> playlist_play_time=<rtsp:/public/foo/foo1.ply, 8, 0>-; time=End-Of-Clip
>
> STREAMING MODULE RESPONSE
> RTSP/1.0 200 OK
> CSeq: 7
> Session: 1234567890
> Range:
> playlist_play_time=<rtsp:/public/foo/foo1.ply, 8, 0>-; npt=65000-
>
>
> The semantics of PLAYLIST_PLAY is almost the same as "PLAY" method as
> defined in RFC2326, with the following exceptions:
> * No RTP-Info header in the response (it is not necessary, since the
> rtp time and seq are continuous -- we are effectively simulating live
> stream).
> * Range type is new, specified in a new format designed specifically
> for playlist support. This "playlist_play_time" is a tupple with the url
> of the playlist file, the clip index, and NPT time inside the clip.
> * The time parameter (see page 34 rfc2326) in Range header takes on
> new reserved tokens:
> * "NOW" means the time to execute the "PLAYLIST_PLAY" requst is right
> now,
> * "End-Of-Clip" means clip boundary
> * "End-Of-Playlist" means end of current playlist
> * In the response to PLAYLIST_PLAY, in the Range header, the
> "time=End-Of-Clip" is replaced with "npt=" parameter, to inform the client
> that End-Of-Clip actually maps to NPT time of 65000. Meaning that the
> presentation of =<rtsp:/public/foo/foo1.ply, 8, 0> will begin at
> NPT=650000. With this information, the client can update the clip banner
> information accordingly. Client also has enough information to derive the
> RTP timestamp corresponding to the first RTP packet for the requested clip
> in the new playlist.
>
> Note the open ended range just means that the end time of the playlist
> session is unknown, in line with Sony requirement.
>
>
> 2. Implementation analysis
>
> In this section we try to compare the merits of the two approaches.
> * SET_PARAMETER approach:
>
> To implement SET_PARAMETER option, the normal scenario is rather
> simple, but problem arises in two areas:

3

```
>       1.  Error handling is more complex.  The following example is
taken
> from section 3.2.4 of Dave's SDD:
>                       STREAMING MODULE ERROR RESPONSE
>                       RTSP/1.0 200 OK
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 24
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_ts: 64385
>                       x-pv-clip_index: 9
>                       x-pv-error_code: Requested clip not found
>
>           Notice that in an RTSP 200 OK response, we are actually
reporting
> an error !
>           In contrast, the same error can be reported, when request is
made
> by PLAYLIST_PLAY, using the 457 code in section 11.3.8 of RFC2326.
Simply:
>                          RTSP/1.0 457
>                          The Range specified a clip that is not found.
>
>           Note that 457 is already implement in Streamer, whilst the
> "x-pv-error_code" fields will have to be thought through and
implemented
> anew -- not a trivial task given that more than 33% of streamer
> development effort has been spent on error handling.
>
>           That is the fundamental reason why the new method approach
will
> end up saving us develeopment  time, we believe.
>
>       2.  SET_PARAMETER approach will be more difficult to adapt to
> potential new requirements. One such new potential feature is the
ability
> to random position to any synch point in any clip across playlists.
For
> Sony it is not possible since PMA client has timing only accurate to
> seconds.
>            In contrast, this new feature can be easily implemented by
> putting a non-zero value in the last field in the
"playlist_play_time"
> tupple. Done !
>
>
> 3. Closing thoughts
>
> Fundamentaly, because we are adding a new feature, i.e., controlling
a new
> type of resource -- playlist resource, it is more natural to extend
> RFC2326  via a method, rather than via some x-pv parameters for
> SET_PARAMETER method.
>
```

Note there is a typo below: it said:
.. whether this has a change to ever become the basis of a playlist
RFC.

I meant:

whether this has a chance to ever become the basis of a playlist RFC.

This Playlist Play method is not in RTSP standard (rfc2326), and could
be
considered an extension to that standard -- in fact, we are considering
submitting a proposal to IETF in the future.

>
> Near the end of the playlist SDD review yesterday, we touched upon
the
> idea of using a new method, in lieu of "SET_PARAMETER", to request a
new
> playlist update or to seek to a new clip within the same playlist.
>
> Although we originally thought that we were too late, schedule-wise,
to
> make such a change, upon researching RTSP RFC more carefully,
thinking
> through it more, and talking to a few people after the meeting, I
have
> realized that, switching to a new method would most likely shorten
the
> overall developement cycle for playlist feature (i.e., 3.4.2
release).
>
> this release will affect the ease of error handling of the protocol
between the
> streamer and the playlist client, the flexiblity of such protocol to
adapt
> to future requirement changes (from Sony or from other customers),
and
> eventualy (perhaps less importantly), whether this has a change to
ever
> become the basis of a playlist RFC.
>
> 1. The two options
>   First, allow me to present the two options in front of us.
> *   SET_PARAMETER
>   This is the approach that we have taken probably since the
> beginning of SDD work. Dave has clearly documented it. Here is the
excerpt
> from page 9 of Dave's latest SDD (version 0.4):
>
>   ...
>   14.   Client application requests a new playlist after a
specified
> amount of time after providing user feedback.
>   15.   PMA server returns a new playlist file and URL.

```
>       16.     Client application calls a new SET_PARAM API on player
> engine and passes it the playlist URL to request streaming from the
new
> playlist.
>       17.     Player engine sends an RTSP SET_PARAMETER command to the
SM
> passing it the playlist URL.
>       18.     SM returns the NPT (normal play time) value at which the
> switch to streaming from the new playlist will occur.
>       19.     The player engine returns this NPT time value back up to
the
> player application so that it can update the display appropriately.
>
> An example SET_PARAMETER to update playlist and jump directly to clip
8,
> at end of current clip, is shown below:
>
>                       CLIENT REQUEST
>                       SET_PARAMETER rtsp://example.com/public/foo
RTSP/1.0
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 40
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_index:  8
>                       x-pv-activate: End-Of-Clip
>
>                       STREAMING MODULE RESPONSE
>                       RTSP/1.0 200 OK
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 24
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_index:  8
>                       x-pv-clip_ts: 65000
>
>
> *     PLAYLIST_PLAY
>
> In stead of using SET_PARAMETER in step 17 above, we follow the
spirit of
> RTSP extension mechanism (see section 1.5 of RFC2326), and use a new
> method, called PLAYLIST_PLAY, to support the new feature for seeking
> across playlists. Here is how it looks, in lieu of the message
exchange
> above:
>
>                       CLIENT REQUEST
>                       PLAYLIST_PLAY rtsp://example.com/public/foo
RTSP/1.0
>                       CSeq: 7
>                       Session: 1234567890
>                       Range:
```

```
> playlist_play_time=<rtsp:/public/foo/fool.ply, 8, 0>-; time=End-Of-
Clip
>
>                       STREAMING MODULE RESPONSE
>                       RTSP/1.0 200 OK
>                       CSeq: 7
>                       Session: 1234567890
>                       Range:
> playlist_play_time=<rtsp:/public/foo/fool.ply, 8, 0>-; npt=65000-
>
>
> The  semantics of PLAYLIST_PLAY is almost the same as "PLAY" method
as
> defined in RFC2326, with the following exceptions:
> *      No RTP-Info header in the response (it is not necessary, since
the
> rtp time and seq are continuous -- we are effectively simulating live
> stream).
> *      Range type is new, specified in a new format designed
specifically
> for playlist support. This "playlist_play_time" is a tupple with the
url
> of the playlist file, the clip index, and  NPT time inside the clip.
> *     The time parameter (see page 34 rfc2326)  in Range header takes
on
> new reserved tokens:
> *      "NOW" means the time to execute the "PLAYLIST_PLAY" requst is
right
> now,
> *     "End-Of-Clip" means clip boundary
> *     "End-Of-Playlist" means end of current playlist
> *      In the response to PLAYLIST_PLAY, in the Range header,  the
> "time=End-Of-Clip" is replaced with "npt=" parameter, to inform the
client
> that End-Of-Clip actually maps to NPT time of 65000. Meaning that the
> presentation of =<rtsp:/public/foo/fool.ply, 8, 0> will begin at
> NPT=650000.  With this information, the client can update the clip
banner
> information accordingly. Client also has enough information to derive
the
> RTP timestamp corresponding to the first RTP packet for the requested
clip
> in the new playlist.
>
> Note the open ended range just means that the end time of the
playlist
> session is unknown, in line with Sony requirement.
>
>
> 2. Implementation  analysis
>
>    In this section we try to compare the merits of the two
approaches.
> *       SET_PARAMETER approach:
>
>     To implement SET_PARAMETER option, the normal scenario is rather
> simple, but problem arises in two areas:
```

7

```
>      1.  Error handling is more complex.  The following example is
taken
> from section 3.2.4 of Dave's SDD:
>                      STREAMING MODULE ERROR RESPONSE
>                      RTSP/1.0 200 OK
>                      CSeq: 7
>                      Session: 1234567890
>                      Content-length: 24
>                      Content-type: text/parameters
>
>                      x-pv-playlist_url: /public/foo/foo1.ply
>                      x-pv-clip_ts: 64385
>                      x-pv-clip_index: 9
>                      x-pv-error_code: Requested clip not found
>
>          Notice that in an RTSP 200 OK response, we are actually
reporting
> an error !
>          In contrast, the same error can be reported, when request is
made
> by PLAYLIST_PLAY, using the 457 code in section 11.3.8 of RFC2326.
Simply:
>                         RTSP/1.0 457
>                         The Range specified a clip that is not found.
>
>          Note that 457 is already implement in Streamer, whilst the
> "x-pv-error_code" fields will have to be thought through and
implemented
> anew -- not a trivial task given that more than 33% of streamer
> development effort has been spent on error handling.
>
>          That is the fundamental reason why the new method approach
will
> end up saving us develeopment  time, we believe.
>
>      2.  SET_PARAMETER approach will be more difficult to adapt to
> potential new requirements. One such new potential feature is the
ability
> to random position to any synch point in any clip across playlists.
For
> Sony it is not possible since PMA client has timing only accurate to
> seconds.
>           In contrast, this new feature can be easily implemented by
> putting a non-zero value in the last field in the
"playlist_play_time"
> tupple. Done !
>
>
> 3. Closing thoughts
>
> Fundamentaly, because we are adding a new feature, i.e., controlling
a new
> type of resource -- playlist resource, it is more natural to extend
> RFC2326  via a method, rather than via some x-pv parameters for
> SET_PARAMETER method.
>
```

8

MMUSIC   Thomas Zeng
Jan 8, 2004

Mapping ICE (Interactive Connectivity Establishment) to RTSP
<draft-zeng-mmusic-map-ice-rtsp-00.txt>

Abstract

This memo describes a mapping from ICE (Interactive Connectivity Establishment) to RTSP for the purpose of Network Address Translator (NAT) traversal for RTSP protocol. In order to become compatible with ICE, the Transport header in RTSP is extended with new syntax elements. This memo presents a few examples RTSP coversations that uses ICE for NAT/firewall traversals.

1. Introduction

ICE protocol is a proposed framework for NAT and firewall traversal.

In [1], the parameters for various ICE messages are defined in generic XML syntax. Each multimedia signalling protocol needs to map these parameters to its own protocol parameters. Section 9 of [1] provides a mapping for SIP (Session Initiation Protocol) based on the SDP Offer/Answer model.

This memo provides a mapping for RTSP (Real-Time Streaming Protocol).

Unlike SIP, RTSP is a multimedia signalling protcol that does not follow the SDP Offer/Answer model defined in RFC3264, for historical reasons.
It is therefore necessary to extend the Transport header in RTSP with new syntax elements in order to fully implement ICE features.

The readers of this memo are expected to have read [1] (especially sections 5 and 9) and have gained a reasonable understand of ICE framework.

RTSP differs from SIP in that RTSP server and RTSP client are almost never deployed behind different NAT/firewalls at the same time. That is, either RTSP server or RTSP client is in the open.

The examples in this memo limit the traversal problem to
1) RTSP server in the open;
2) RTSP client in the open.

In such cases, TURN services are not required for connectivity establishment between RTSP server and client.

2. Mapping ICE to RTSP

2.1 How Does ICE Work For RTSP: an overview

The key assumption in ICE is that a signalling entity cannot know,
apriori,
whether the peer it wishes to communicate with is connected to one
or
all of the address realms it is in. Therefore, in order to
communicate, it has to try them all, and choose the best one that
works. This assumption is true for RTSP.

As described in figure 1, section 2 of [1], in terms of signalling
model for RTSP, the initiator is the RTSP
client, the responder is the RTSP server, the initiate message is a
SETUP message, and the accept message is a SETUP response. The
modify
message is a SETUP message, and the modify acceptance message is a
SETUP response.

It is also an option to treat the DESCRIBE response from RTSP server
to RTSP client as another initiate message in the ICE context. For
RTSP, DESCRIBE response normally carries the session description
in SDP format. It is in this SDP that the RTSP server may use the
SDP extension in section 9 of [1] to inform its client of the
addresses, ports and associated parameters (e.g., user name,
password)
that the server has discovered. However, since not all RTSP
sessions begin with DESCRIBE (many rely on ftp or HTTP protocols to
obtain session descriptions out of band), in the rest of this memo,
we will only consider SETUP as the initiate message, even though
starting ICE process with DESCRIBE response can save up to one
round of ICE negotiations.

Here is how ICE would work with RTSP.

Before the RTSP client establishes a session, it obtains as many IP
address and port combinations in as many address realms as it can.
Any protocol that provides a
client with an IP address and port on which the RTSP client
can receive traffic
can be used. These include STUN and even VPN. The RTSP
client also uses any local interface addresses. A dual-stack v4/v6
client will obtain both a v6 and a v4 address/port. The only
requirement is that, across all of these addresses, the RTSP client
can be certain that at least one of them will work for any
responder it might communicate with. This is guaranteed by:

1) The assumption that the RTSP client and server are separated
   by at most one level of NAT/firewall;
2) The assumption that co-located STUN servers can be installed
   on the media ports in each protocol entity.

The RTSP client then makes a STUN server available on each of the
address/port combinations it has obtained. This STUN server is

running locally, on the initiator. All of these addresses are placed
into the Tranport header of the SETUP request and they are ordered
in
terms of preference given in [1]. The SETUP request also conveys
the STUN username and password which are required to gain access to
the STUN server on each address/port combination. Tranport header
extensions are described in the next section to convey username and
password.

The initiate message -- the SETUP request,
is sent to the responder(normally RTSP server)
via the RTSP connection, preferably using
a secure protocol such as TLS.

Once the RTSP server receives the SETUP request,
it sends STUN requests to each alternate address/

port in the Transport. These STUN requests include the
username and password obtained from the initiate message.
The STUN requests serve two purposes. The first
is to check for connectivity. If a response is received, the
RTSP server knows that it can reach the client at that address. The
second purpose is to obtain more addresses at which the RTSP server
can be contacted. If the client is behind a NAT,
the RTSP server may discover another address through the STUN
responses. In its accept message -- 200 OK Setup response, the
RTSP server includes all
addresses that it can unilaterally determine (just as the client
did), in addition to any that were discovered using the STUN
messages
to the RTSP client.

When the accept message arrives at the RTSP client, the client
performs a similar operation. Using STUN, it checks connectivity to
each of the addresses in the accept message. Through the STUN
responses, it may learn of additional addresses that it can use to
receive media. If it does learn any new address, the clinet
generates
a modify message to pass
this address to the RTSP server. For RTSP, modify message is
re-SETUP request.
The RTSP server processes the re-SETUP request as a "ICE Modify"
message
and sends a "200 OK" SETUP response as the "ICE Modify response"
message.

At this point, ICE process is complete, or else connection
cannot be established.

2.1 Extending RTSP Transport Header Syntax

In order to convey username and password used to access colocated
STUN servers, it is necessary to extend the RTSP Transport header
definitions in [4].

```
   Transport                  =  "Transport" ":" 1#transport-spec
   transport-spec             =  transport-id *parameter
   transport-id               =  transport-protocol "/" profile
                                 ["/" lower-transport]
                           ; no LWS is allowed inside transport-id
   transport-protocol         =  "RTP" / token
   profile                    =  "AVP" / token
   lower-transport            =  "TCP" / "UDP" / token
   parameter                  =  ";" ( "unicast" / "multicast" )
                                 ...
                                 ...
                             /   ";" "dest_addr" "=" addr-list
                             /   ";" "src_addr" "=" addr-list
                             /   ";" "username" "=" non-ws-string
                             /   ";" "password" "=" non-ws-string
                            ; the above two are new parameters for ICE
                             /   ";" trn-parameter-extension
```

3. Terminology

Several new terms are introduced in [1] and elaborated in this memo:

Session Initiator: A software entity that, at the request of a user, tries to establish communications with another entity, called the session responder. A session initiator is also called an initiator. In RTSP context, initiator is normally the RTSP client.

Initiator: Another term for a session initiator.

Session Responder: A software entity that receives a request for establishment of communications from the session initiator, and either accepts or declines the request. A session responder is also called a responder. In RTSP context, a session responder is normally the RTSP server.

Responder: Another term for a session responder.

Initiate Message: The signaling message used by an initiator to establish communications. It contains capabilities and other information needed by the responder to send media to the initiator.

Accept Message: The signaling message used by a responder to agree to communications. It contains capabilities and other information needed by the initiator to send media to the responder.

Modify Message: The signaling message used by either an initiator or responder to change the capability and other information needed by the peer for sending media.

Modify Acceptance Message: The signaling message used by a client to

agree to the changes proposed in a modify message, and to present
      the capability or other information needed by its peer for
sending
      media.

   Protocol Entity: either side of the media stream. For RTSP, a
      protocol entity is either the RTSP server or the RTSP client.

   Terminate Message The signaling message used by a client to
terminate
      the session and associated media streams.

   Transport Address: The combination of an IP address and port.

   Local Transport Address: A local transport address is transport
      address that has been allocated from the operating system on the
      host. This includes transport addresses obtained through VPNs,
and
      also transport addresses obtained through RSIP (which lives at
the
      operating system level). Transport addresses are typically
      obtained by binding to an interface.

   Derived Transport Address: A derived transport address is a
transport
      address which is associated with, but different from, a local
      transport address. The derived transport address is associated
      with the local transport address in that packets sent to the
      derived transport address are received on the socket bound to
that
      local transport address. Derived addresses are obtained using
      protocols like STUN and TURN, and more generally, any UNSAF
      protocol [11].

4. Example RTSP Conversations

The examples below follow the RTSP ABNF rules in [4]. It is worth noting that some of the syntax elements, such as "dest_addr" and "src_addr", are new to [4], and were not in RFC2326.

4.1 Case 1: RTSP server is in the open; client behind cone NAT

The following sample RTSP conversation describes how ICE traverses a "cone" NAT on behalf of an RTSP client behind NAT. In this example, RTSP server is in the public address realm, which is true for most RTSP serive deployment to-date.

Recall from [5] the definition of cone NATs:

Full Cone: A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

Restricted Cone: A restricted cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

We assume that the RTSP clinet behind this cone NAT obtains its external IP (i.e., 24.2.1.1) and port apriori, using a public STUN server. The RTSP client's first SETUP request includes two choices of addresses for RTP/RTCP ports, as shown by the relevant RTSP conversation below:

```
    C->S  SETUP rtsp://foo.com/test.wav/streamid=0 RTSP/1.0
             Transport: RTP/AVP/UDP;unicast;src_addr="172.16.1.1:6970"/
                        "172.16.1.1:6971"; username="foo";
password="x",
                        RTP/AVP/UDP;unicast;src_addr="24.2.1.1:9970"/
                        "24.2.1.1:9980"; username="server";
password="s"
             CSeq: 2

    S->C  RTSP/1.0 200 OK
             Transport: RTP/AVP/UDP;unicast;dest_addr="24.2.1.1:9970"/
                        "24.2.1.1:9980"; src_addr="24.2.8.8:5540"/
                        "24.2.8.8:5541"; username="client";
password="c"
             CSeq: 2
             Session: 2034820394
```

Comments: in the first SETUP request message, there are two tranport specifications, separated by a comma as per [4].
The first transport uses local address and port, while the second uses STUN

discovered public address and port. In the 200 OK response, the
presence
of "dest_addr" parameter indicates that RTSP server has completed
its ICE process after successful STUN bindings.

Finally RTSP client performs STUN bindings against the RTSP server
using the "src_addr", username and password in the SETUP request, and
receives STUN responses.

In this example, connectivity is established in only one round of
ICE
negotiation, thanks to the fact that STUN binding is performed
approri.
A nice benefit is that RTSP conversational
delay is not increased by much. But connectivity may not always be
established in one SETUP / Response cycle.
In the case of symetric NAT, STUN binding must be done
during RTSP conversations, not before, as shown by the next example.

4.2 Case 2: RTSP server is in the open; client behind Symetric NAT

In this case, obtaining external IP address and port
appriori is of no value,
given the symetric nature of the NAT. Therefore, the RTSP client does
not list any public address in its first SETUP request.

```
  C->S  SETUP rtsp://foo.com/test.wav/streamid=0 RTSP/1.0
           Transport: RTP/AVP/UDP;unicast;src_addr="172.16.1.1:6970"/
                      "172.16.1.1:6971"; username="foo"; password="x"
           CSeq: 2
```

/* RTSP server cannot reach "172.16.1.1". The server's STUN binding
 request will timeout, and it then sends the following response.
 The lessen here is that STUN binding timeout should be set to
 a fairly short value so as to minimize the impact on RTSP delay. */

```
  S->C  RTSP/1.0 200 OK
           Transport: RTP/AVP/UDP;unicast; src_addr="24.2.8.8:5540"/
                      "24.2.8.8:5541"; username="client";
password="c"
           CSeq: 2
           Session: 2034820394
```

/* RTSP client now performs STUN bindings and finds its external
address/port pair as, say, "24.2.1.1:6970"/"24.2.1:6971",
it then sends re-SETUP as ICE modify message: */

```
  C->S  SETUP rtsp://foo.com/test.wav/streamid=0 RTSP/1.0
           Transport: RTP/AVP/UDP;unicast;src_addr="172.16.1.1:6970"/
                      "172.16.1.1:6971"; username="foo"; password="x"
                      RTP/AVP/UDP;unicast;src_addr="24.2.1.1:6970"/
                      "24.2.1.1:6971"; username="server";
password="s"
           CSeq: 3
```

```
             Session: 2034820394
  /* RTSP server can reach 24.2.1.1. So it sends the following 200 OK:
*/

    S->C  RTSP/1.0 200 OK
             Transport: RTP/AVP/UDP;unicast;dest_addr="24.2.1.1:6970"/
                        "24.2.1.1:6971"; src_addr="24.2.8.8:5540"/
                        "24.2.8.8:5541"; username="client";
password="c"
             CSeq: 3
             Session: 2034820394
```

  Comments: RTSP SETUP delay has been increased in this case by two
factors,
  when compared to case 1:
     1) STUN timeout after the first SETUP request is received by RTSP
        server.
     2) Additoinal SETUP/response round trip.

Case 3: RTSP client is in the open, RTSP server is behind symetric NAT

  In this scenario, client has only one address to include in its
  SETUP request.

```
    C->S  SETUP rtsp://foo.com/test.wav/streamid=0 RTSP/1.0
             Transport: RTP/AVP/UDP;unicast;src_addr="24.2.1.1:6970"/
                        "24.2.1.1:6971"; username="server";
                        password="x"
             CSeq: 2

  /* RTSP server's STUN packets can reach "24.2.1.1" and discover its
   own external IP/port as 24.2.8.8/5540 and 24.2.8.8/5541 (RTCP). */


    S->C  RTSP/1.0 200 OK
             Transport: RTP/AVP/UDP;unicast; dest_addr="24.2.1.1:6970"/
                        "24.2.1.1:6971"; src_addr="24.2.8.8:5540"/
                        "24.2.8.8:5541"; username="client";
password="c"
             CSeq: 2
             Session: 2034820394
```

  Here no additional RTSP message exchange is needed.

5. Security Considerations

  The sections titled "security considerations" in [1] and [4] covers
  all the security considerations relevant to this memo. No additional
  consideration is deemed necessary.

Normative References

[1] Rosenberg, J., " Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal ", draft-ietf-mmusic-ice-00, October 2003.

[2] Rosenberg, J., Weinberger, J., Huitema, C. and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.

[3] Camarillo, G. and J. Rosenberg, "The Alternative Semantics for the Session Description Protocol Grouping Framework", draft-camarillo-mmusic-alt-01 (work in progress), June 2003.

[4] H. Schulzrinne, et. al., "Real Time Streaming Protocol (RTSP)", draft-ietf-mmusic-rfc2326bis-05.txt, Oct 2003

[5] Westlunder, M. and Zeng, T., "How to make Real-Time Streaming Protocol (RTSP) traverse Network Address Translators (NAT) and interact with Firewalls", draft-ietf-mmusic-rtsp-nat-01.txt, May 2003

[6] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines", RFC 3235, January 2002.

[7] Borella, M., Lo, J., Grabelsky, D. and G. Montenegro, "Realm Specific IP: Framework", RFC 3102, October 2001.

Author's Address

Thomas Zeng
PV Network Solutions,
10350 Science Center Dr., Suite 200
San Diego, CA92127
US

Phone: +1 858 731 5465
EMail: zeng@pv.com

PacketVideo Network
T. Zeng
P. Greg

Using RTSP Announce to signal End Of Stream
draft-zeng-mmusic-01

Status of this Memo

Abstract

This document describes an extension RTSP method, ANNOUNCE, which extends the core RTSP protocol to enable one RTSP protocol entity to push various types of information to the other end. Such information may include session description or end of stream events.

The receiver of this RTSP request is expected to reply with 200 OK response.

Since ANNOUNCE represents an extension to the core RTSP protocol, a feature tag,
"method.announce", is defined to facilitate capability negotiation using RTSP extension mechanism [RTSP_NEW].

Changes From Previous Version:

1. Replaced END_OF_STREAM method with "Announce" method
2. Added use case for "announce" method to convey SDP

1. Motivation

In the core RTSP protocol [RTSP_NEW], ANNOUNCE method is taken out. There are use cases that require ANNOUNCE method to convey session description. There is a need to make this functionality still available but in a way consistent to the extension mechanism in [RTSP_NEW].

Meanwhile, there is also a need to signal end of stream event from server to client, as detailed below.

In the core RTSP protocol [RTSP_NEW], an RTSP client relies on the media transport mechanism to signal end of stream.

When the media transport mechanism happens to be RTP over UDP, this is carried out by RTCP BYE packet [RTP_NEW]. In practice, there are some drawbacks with this approach:

1. When the server sends an RTCP BYE packet with its SSRC, the server is giving up
the SSRC (see section 8.2 in [RTP_NEW]). The server would be required to
switch to a new SSRC on a subsequent PLAY of the same media stream.

Since server's SSRC is only communicated in the Transport
header
of SETUP
response, the server would not have an opportunity to send a
new
value to
the player, and the client would have to discover the SSRC from
the incoming RTP packets.

2. RTCP BYE packet method does not offer a simple, gurranteed
method of delievering
an end-of-stream announcement.

3. RTCP BYE packet method does not offer the option to have a
single
aggregate
end-of-stream announcemnt for all media streams in the RTSP
session.

4. Section 6.3.7 of RFC3550 stipulates that an RTP sender cannot
send RTCP BYE
before leaving the RTP session if it has not already sent at
least one RTP or RTCP packet. This is a problem under
error conditions. Consider the case
where an RTP session has just started (i.e., RTSP PLAY has
been
successfully acknowledged with an RTSP 200 OK response), and
the
sender attempts to
retrieve media frames from its media source. The media source
fails to provide any media frame due to its internal error such
as file corruption. The sender should inform its receiver(s)
but it cannot send BYE packets.

The motivation to solve the above issues is particularly high for
unicast streaming applications that use RTSP over TCP in the control
plane, and RTP over UDP in the media transport.

There is also the desire to have an EOS (End Of Stream)
signalling mechanism
for non-RTP delivery. One such delivery is MPEG2 transport streams
used in the cable TV environment. In non-IP delivery environments,
the transport typically remains allocated even if no media is being
delivered. This
means that a client cannot watch for the server to close the
transport to signal the end of stream. Meanwhile, watching for the
incoming media to stop is unreliable. Short
timeouts can trigger a false end of media detection if the
media flow is temporarily delayed. Long timeouts introduce
unacceptable latencies. Clients are unable to distinguish
between a normal end of stream and an error condition that
resulted in the media delivery stopping.

We note that using TEARDOWN from server to client is not
appropriate because:
1. TEARDOWN is currently not allowed from server to
client [RTSP_NEW];

2. Even if TEARDOWN is made available in server to client
   direction,
   the definition of TEARDOWN requires that, if the request
   URI is
   aggregate, that the session must be de-allocated by the
   server.
   There are RTSP applications that use SET_PARAMETER from
   client to
   server as the means to report session QoS statistics, but if
   server uses TEARDOWN on aggregate URL to signal end of stream,
   the client can no longer use SET_PARAMETER with a
   session header.
3. In general, RTSP, being a client-server protocol,
   should let client, not server to control session state. But
   TEARDOWN
   on aggregate URL will change session from PLAYING state
   to INIT state.

We note that using PAUSE from server to client is not appropriate either, because PAUSE will change the state of the RTSP session.

This memo proposes to define ANNOUNCE method to singal end of stream. ANNOUNCE method is a server to client RTSP request originally defined in RFC2326 but excluded from the core RTSP protocol due to lack of support [RTSP_NEW].

This memo defines ANNOUNCE as an extension to the core RTSP protocol [RTSP_NEW]. It presents ANNOUNCE method as a general mechanism for RTSP server to signal to its clients various events including end of stream events or session description update events, among others. This memo will discuss the general usage of ANNOUNCE as well.

3. The Definition of ANNOUNCE method

[RTSP_NEW] clearly defines the mechanism to extend the core RTSP protocol. Following that mechanism, a feature tag is used to identify ANNOUCE method as an extension to the core RTSP protocol.

ANNOUNCE method is a RTSP request that can be sent in both directions, either from client to server or server to client. When server intends to send ANNOUNCE to client, it must have already a transport mechanism available to reach the client, because the RTSP client is not required to keep a persistent connection with the RTSP server. It is beyond the scope of this memo to define the exact means for server to reach client. It is suffice to say that if the client intends to receive server's ANNOUNCE methods, it must keep the RTSP connection open, or inform the server on how to reach it without a persistent RTSP connection.

The purpose of RTSP ANNOUNCE method is to allow server or client to push different types of information to the other end. Such information could be session description information, end of stream event or some type of error events, among others.

Here is an example in which an RTSP server announces end of stream event for a media stream using a non-aggregate URI.

```
 S->C: ANNOUNCE rtsp://foo.com/bar.avi/streamid=0 RTSP/1.0
       CSeq: 10
       Session: 12345678
       Notice: End-Of-Stream
       Reason: 2000 End of range reached
       Range: npt=0-200
       RTP-Info: url=//foo.com/bar.avi/streamid=0;seq=45102

 C->S: RTSP/1.0 200 OK
       CSeq: 10
       Session: 12345678
```

3.1 Normative definitions

"ANNOUNCE" is an "extension-method" in the ABNF in section 16.2 "RTSP Protocol Definitions" in [RTSP_NEW].

The request-URI of an ANNOUNCE request can be either aggregate or non-aggregate URI.

An ANNOUNCE request must include "CSeq" header. It MAY include the following optional headers:

  "Range", "Session", "Reason", "RTP-Info"

An ANNOUNCE request MAY include entity body, in which case it MUST follow the rules for entity body defined in section 8.2 of [RTSP_NEW].

ANNOUNCE does NOT change RTSP session state.

3.2 New header for ANNOUNCE to signal end of stream

When ANNOUNCE is used for RTSP server to signal end of stream, it MUST include "CSeq", "Range" and "Session" headers.
It SHOULD include "RTP-Info" header.
The RTP-Info in server's END_OF_STREAM request
is used to indicate the sequence number of
the ending RTP packet for each media stream.

It MAY include a new RTSP header, the "Reason" header, defined in ABNF as:

```
       Reason     =  "Reason" ":"   Reason-Phrase CRLF
```

where:
- -- "Reason-Phrase" is a parameter defined in section 7.1.1 of [RTSP_NEW].

The purpose is to allow the server to explain why stream has ended.

Note that the server is free to use any text as "Reason-Phrase". In certain applications, the client
may display the text, in its entirety, to end user to improve user friendliness.

3.3 Limitations on serve to client "ANNOUNCE" requests

Server to client ANNOUNCE method is issued only if the server has the means to contact the client when it has information to push. This may not be possible if the RTSP connection between server and client is not persistent. In such cases, the server will simply skip the sending of ANNOUNCE requests. That is to say, the server will not queue up the ANNOUNCE requests to be sent when client eventually connects. Such a queue would un-necessarily complicate server implementations.

4. Feature tag

The ANNOUNCE method is represented by a feature tag (see [RTSP_NEW]
for how to use feature tags in capability negotiations).

The feature tag is:

```
      method.announce
```

Implementations claiming "method.announce" feature tag MUST support the new "Reason" header defined in previsous section.

5. Use Cases

This section presents some use cases of the ANNOUNCE method.

5.1 Client Announcing SDP To Server For Recording

This use case is the same as the first RTSP exchange presented in section 14.6 in RFC2326, with capability exchanged added via OPTIONS method.

The conference participant client C asks the media server M to record
the audio and video portions of a meeting. The client uses the ANNOUNCE method to provide meta-information about the recorded session to the server; but first, it makes sure that "ANNOUNCE" feature is supported by the server.

```
  C->M: OPTIONS * RTSP/1.0
        Require: method.announce
        CSeq: 1

  M->C: RTSP/1.0 200 OK
        CSeq: 1
        Supported: method.announce
        Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE,
RECORD,ANNOUNCE
```

```
      C->M: ANNOUNCE rtsp://server.example.com/meeting RTSP/1.0
            CSeq: 90
            Content-Type: application/sdp
            Content-Length: 121

            v=0
            o=camera1 3080117314 3080118787 IN IP4 195.27.192.36
            s=IETF Meeting, Munich - 1
            i=The thirty-ninth IETF meeting will be held in Munich,
Germany
            u=http://www.ietf.org/meetings/Munich.html
            e=IETF Channel 1 <ietf39-mbone@uni-koeln.de>
            p=IETF Channel 1 +49-172-2312 451
            c=IN IP4 224.0.1.11/127
            t=3080271600 3080703600
            a=tool:sdr v2.4a6
            a=type:test
            m=audio 21010 RTP/AVP 5
            c=IN IP4 224.0.1.11/127
            a=ptime:40
            m=video 61010 RTP/AVP 31
            c=IN IP4 224.0.1.12/127

      M->C: RTSP/1.0 200 OK
            Supported: method.announce
            CSeq: 90
```

5.2 Server Announcing End Of Stream

In this example, the server announces the END_OF_STREAM event to client for one live media stream, because upstream source terminates the stream after 200 seconds. The fact that the stream has played for 200 seconds is communicated by the Range header in the ANNOUNCE request.

```
      C->S: PLAY rtsp://foo.com/bar.avi/streamid=0 RTSP/1.0
            Supported: method.annonce
            CSeq: 10
            Session: 12345678
            Range: 0-200

      S->C: RTSP/1.0 200 OK
            Supported: method.announce
            CSeq: 10
            Session: 12345678
            RTP-Info: url=//foo.com/bar.avi/streamid=0;seq=0; rtptime=0

      S->C: ANNOUNCE rtsp://foo.com/bar.avi/streamid=0 RTSP/1.0
            CSeq: 123
            Session: 12345678
            Range: npt=0-200
            RTP-Info: url=//foo.com/bar.avi/streamid=0;seq=45102
            Reason: End of range reached

      C->S: RTSP/1.0 200 OK
            CSeq: 123
```

```
Session: 12345678
```

From the ANNOUNCE request, the client will learn that the server has completed the stream as requested (as indicated by the "End of range reached" Reason).

From the two RTP-Info headers, one in PLAY response, one in ANNOUNCE requst, the client can derive the total number of RTP packets that the server has sent. From the npt field in the Range header, the client knows the presentation time that this stream has covered, from the server's point of view.

6. Security Considerations

Because there is only one new TEXT header, "Reason", added by the extension RTSP method, the security considerations outlined in [RTSP_NEW] apply here as well.

7. IANA Considerations

A new method name, and its associated feature tag need to be registered with IANA.

-- Method name: ANNOUNCE

-- feature tag: method.announce

Normative References

[RTSP_NEW] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., "Real Time Streaming Protocol", draft-ietf-mmusic-rfc2326bis-04.txt

[RTP_NEW] RFC3550 "Real-time Transport Protocol", July 2003

Author Addresses

Thomas Zeng
PacketVideo Network Solutions
9605 Scranton Road, Suite 400
San Diego, CA 92121
email: zeng@pvnetsolutions.com

P. Greg Sherwood
PacketVideo Device Solutions.
10350 Science Center Dr., Suite 210
San Diego, CA 92121
email: sherwood@pv.com

PacketVideo Network
T. Zeng
P. Greg

Using RTSP Announce to signal End Of Stream
draft-zeng-mmusic-01

Status of this Memo

Abstract

This document describes an extension RTSP method, ANNOUNCE, which extends the core RTSP protocol to enable one RTSP protocol entity to push various types of information to the other end. Such information may include session description or end of stream events.

The receiver of this RTSP request is expected to reply with 200 OK response.

Since ANNOUNCE represents an extension to the core RTSP protocol, a feature tag,
"method.announce", is defined to facilitate capability negotiation using RTSP extension mechanism [RTSP_NEW].

Changes From Previous Version:
1. Replaced END_OF_STREAM method with "Announce" method
2. Added use case for "announce" method to convey SDP

1. Motivation

In the core RTSP protocol [RTSP_NEW], ANNOUNCE method is taken out. There are use cases that require ANNOUNCE method to convey session description. There is a need to make this functionality still available but in a way consistent to the extension mechanism in [RTSP_NEW].

Meanwhile, there is also a need to signal end of stream event from server to client, as detailed below.

In the core RTSP protocol [RTSP_NEW], an RTSP client relies on the media transport mechanism to signal end of stream.

When the media transport mechanism happens to be RTP over UDP, this is carried out by RTCP BYE packet [RTP_NEW]. In practice, there are some drawbacks with this approach:

1. When the server sends an RTCP BYE packet with its SSRC, the server is giving up
   the SSRC (see section 8.2 in [RTP_NEW]). The server would be required to
   switch to a new SSRC on a subsequent PLAY of the same media stream.

Since server's SSRC is only communicated in the Transport
header
of SETUP
response, the server would not have an opportunity to send a
new
value to
the player, and the client would have to discover the SSRC from
the incoming RTP packets.

2. RTCP BYE packet method does not offer a simple, gurranteed
method of delievering
an end-of-stream announcement.

3. RTCP BYE packet method does not offer the option to have a
single
aggregate
end-of-stream announcemnt for all media streams in the RTSP
session.

4. Section 6.3.7 of RFC3550 stipulates that an RTP sender cannot
send RTCP BYE
before leaving the RTP session if it has not already sent at
least one RTP or RTCP packet. This is a problem under
error conditions. Consider the case
where an RTP session has just started (i.e., RTSP PLAY has
been
successfully acknowledged with an RTSP 200 OK response), and
the
sender attempts to
retrieve media frames from its media source. The media source
fails to provide any media frame due to its internal error such
as file corruption. The sender should inform its receiver(s)
but it cannot send BYE packets.

The motivation to solve the above issues is particularly high for
unicast streaming applications that use RTSP over TCP in the control
plane, and RTP over UDP in the media transport.

There is also the desire to have an EOS (End Of Stream)
signalling mechanism
for non-RTP delivery. One such delivery is MPEG2 transport streams
used in the cable TV environment. In non-IP delivery environments,
the transport typically remains allocated even if no media is being
delivered. This
means that a client cannot watch for the server to close the
transport to signal the end of stream. Meanwhile, watching for the
incoming media to stop is unreliable. Short
timeouts can trigger a false end of media detection if the
media flow is temporarily delayed. Long timeouts introduce
unacceptable latencies. Clients are unable to distinguish
between a normal end of stream and an error condition that
resulted in the media delivery stopping.

We note that using TEARDOWN from server to client is not
appropriate because:
1. TEARDOWN is currently not allowed from server to
client [RTSP_NEW];

2. Even if TEARDOWN is made available in server to client direction,
   the definition of TEARDOWN requires that, if the request URI is
   aggregate, that the session must be de-allocated by the server.
   There are RTSP applications that use SET_PARAMETER from client to
   server as the means to report session QoS statistics, but if
   server uses TEARDOWN on aggregate URL to signal end of stream,
   the client can no longer use SET_PARAMETER with a session header.
3. In general, RTSP, being a client-server protocol,
   should let client, not server to control session state. But TEARDOWN
   on aggregate URL will change session from PLAYING state
   to INIT state.

We note that using PAUSE from server to client is not appropriate either, because PAUSE will change the state of the RTSP session.

This memo proposes to define ANNOUNCE method to singal end of stream. ANNOUNCE method is a server to client RTSP request originally defined in RFC2326 but excluded from the core RTSP protocol due to lack of support [RTSP_NEW].

This memo defines ANNOUNCE as an extension to the core RTSP protocol [RTSP_NEW]. It presents ANNOUNCE method as a general mechanism for RTSP server to signal to its clients various events including end of stream events or session description update events, among others. This memo will discuss the general usage of ANNOUNCE as well.

3. The Definition of ANNOUNCE method

[RTSP_NEW] clearly defines the mechanism to extend the core RTSP protocol. Following that mechanism, a feature tag is used to identify ANNOUCE method as an extension to the core RTSP protocol.

ANNOUNCE method is a RTSP request that can be sent in both directions, either from client to server or server to client. When server intends to send ANNOUNCE to client, it must have already a transport mechanism available to reach the client, because the RTSP client is not required to keep a persistent connection with the RTSP server. It is beyond the scope of this memo to define the exact means for server to reach client. It is suffice to say that if the client intends to receive server's ANNOUNCE methods, it must keep the RTSP connection open, or inform the server on how to reach it without a persistent RTSP connection.

The purpose of RTSP ANNOUNCE method is to allow server or client to push different types of information to the other end. Such information could be session description information, end of stream event or some type of error events, among others.

Here is an example in which an RTSP server announces end of stream event for a media stream using a non-aggregate URI.

```
 S->C: ANNOUNCE rtsp://foo.com/bar.avi/streamid=0 RTSP/1.0
       CSeq: 10
       Session: 12345678
       Notice: End-Of-Stream
       Reason: 2000 End of range reached
       Range: npt=0-200
       RTP-Info: url=//foo.com/bar.avi/streamid=0;seq=45102

 C->S: RTSP/1.0 200 OK
       CSeq: 10
       Session: 12345678
```

3.1 Normative definitions

"ANNOUNCE" is an "extension-method" in the ABNF in section 16.2 "RTSP Protocol Definitions" in [RTSP_NEW].

The request-URI of an ANNOUNCE request can be either aggregate or non-aggregate URI.

An ANNOUNCE request must include "CSeq" header. It MAY include the following optional headers:

  "Range", "Session", "Reason", "RTP-Info"

An ANNOUNCE request MAY include entity body, in which case it MUST follow the rules for entity body defined in section 8.2 of [RTSP_NEW].

ANNOUNCE does NOT change RTSP session state.

3.2 New header for ANNOUNCE to signal end of stream

When ANNOUNCE is used for RTSP server to signal end of stream, it MUST include "CSeq", "Range" and "Session" headers.
It SHOULD include "RTP-Info" header.
The RTP-Info in server's END_OF_STREAM request
is used to indicate the sequence number of
the ending RTP packet for each media stream.

It MAY include a new RTSP header, the "Reason" header, defined in ABNF as:

```
       Reason     =  "Reason" ":"   Reason-Phrase CRLF
```

where:
    -- "Reason-Phrase" is a parameter defined in section 7.1.1 of [RTSP_NEW].

The purpose is to allow the server to explain why stream has ended.

Note that the server is free to use any text as "Reason-Phrase". In certain applications, the client
may display the text, in its entirety, to end user to improve user friendliness.

3.3 Limitations on serve to client "ANNOUNCE" requests

Server to client ANNOUNCE method is issued only if the server has the means to contact the client when it has information to push. This may not be possible if the RTSP connection between server and client is not persistent. In such cases, the server will simply skip the sending of ANNOUNCE requests. That is to say, the server will not queue up the ANNOUNCE requests to be sent when client eventually connects. Such a queue would un-necessarily complicate server implementations.

4. Feature tag

The ANNOUNCE method is represented by a feature tag (see [RTSP_NEW]
for how to use feature tags in capability negotiations).

The feature tag is:

```
method.announce
```

Implementations claiming "method.announce" feature tag MUST support the new "Reason" header defined in previsous section.

5. Use Cases

This section presents some use cases of the ANNOUNCE method.

5.1 Client Announcing SDP To Server For Recording

This use case is the same as the first RTSP exchange presented in section 14.6 in RFC2326, with capability exchanged added via OPTIONS method.

The conference participant client C asks the media server M to record
the audio and video portions of a meeting. The client uses the ANNOUNCE method to provide meta-information about the recorded session to the server; but first, it makes sure that "ANNOUNCE" feature is supported by the server.

```
  C->M: OPTIONS * RTSP/1.0
        Require: method.announce
        CSeq: 1

  M->C: RTSP/1.0 200 OK
        CSeq: 1
        Supported: method.announce
        Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE,
RECORD,ANNOUNCE
```

```
      C->M: ANNOUNCE rtsp://server.example.com/meeting RTSP/1.0
            CSeq: 90
            Content-Type: application/sdp
            Content-Length: 121

            v=0
            o=camera1 3080117314 3080118787 IN IP4 195.27.192.36
            s=IETF Meeting, Munich - 1
            i=The thirty-ninth IETF meeting will be held in Munich,
Germany
            u=http://www.ietf.org/meetings/Munich.html
            e=IETF Channel 1 <ietf39-mbone@uni-koeln.de>
            p=IETF Channel 1 +49-172-2312 451
            c=IN IP4 224.0.1.11/127
            t=3080271600 3080703600
            a=tool:sdr v2.4a6
            a=type:test
            m=audio 21010 RTP/AVP 5
            c=IN IP4 224.0.1.11/127
            a=ptime:40
            m=video 61010 RTP/AVP 31
            c=IN IP4 224.0.1.12/127

      M->C: RTSP/1.0 200 OK
            Supported: method.announce
            CSeq: 90
```

   5.2 Server Announcing End Of Stream

   In this example, the server announces the END_OF_STREAM event to
client for
   one live media stream, because upstream source terminates the stream
   after 200 seconds. The fact that the stream has played for 200
seconds
   is communicated by the Range header in the ANNOUNCE request.

```
      C->S: PLAY rtsp://foo.com/bar.avi/streamid=0 RTSP/1.0
            Supported: method.annonce
            CSeq: 10
            Session: 12345678
            Range: 0-200

      S->C: RTSP/1.0 200 OK
            Supported: method.announce
            CSeq: 10
            Session: 12345678
            RTP-Info: url=//foo.com/bar.avi/streamid=0;seq=0; rtptime=0

      S->C: ANNOUNCE rtsp://foo.com/bar.avi/streamid=0 RTSP/1.0
            CSeq: 123
            Session: 12345678
            Range: npt=0-200
            RTP-Info: url=//foo.com/bar.avi/streamid=0;seq=45102
            Reason: End of range reached

      C->S: RTSP/1.0 200 OK
            CSeq: 123
```

```
         Session: 12345678
```

From the ANNOUNCE request, the client will learn that the server has completed the stream as requested (as indicated by the "End of range reached" Reason).

From the two RTP-Info headers, one in PLAY response, one in ANNOUNCE requst, the client can derive the total number of RTP packets that the server has sent. From the npt field in the Range header, the client knows the presentation time that this stream has covered, from the server's point of view.

6. Security Considerations

Because there is only one new TEXT header, "Reason", added by the extension RTSP method, the security considerations outlined in [RTSP_NEW] apply here as well.

7. IANA Considerations

A new method name, and its associated feature tag need to be registered with IANA.

-- Method name: ANNOUNCE

-- feature tag: method.announce

Normative References

[RTSP_NEW] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., "Real Time Streaming Protocol", draft-ietf-mmusic-rfc2326bis-04.txt

[RTP_NEW] RFC3550 "Real-time Transport Protocol", July 2003

Author Addresses

Thomas Zeng
PacketVideo Network Solutions
9605 Scranton Road, Suite 400
San Diego, CA 92121
email: zeng@pvnetsolutions.com

P. Greg Sherwood
PacketVideo Device Solutions.
10350 Science Center Dr., Suite 210
San Diego, CA 92121
email: sherwood@pv.com

Oct 20, 2003

RAFT: RTSP Authenticatable Firewall Tranversal Protocol
draft-zeng-mmusic-raft-00

Status of this Memo

Abstract

This document describes a simple firewall traversal protocol for the RTP data stream of an RTSP session.
This protocol solves not only solves the firewall traversal problem, but also provides authentication to guard against DoS (Denial of Service) attacks in applications where the RTP packet receiver has a different IP address than the RTSP client.

The protocol assumes that the RTSP server is in the public domain network, and that RTSP clients may or may not reside behind firewalls or NATs - Native Address Translators. For simplicity we use "firewall" to refer to any middle box, firewalls and NATs alike, that masqurades the IP address and/or port. More over, we assume that the media transport is using RTP over UDP.

From the client's RTP receive socket, the client sends probing UDP packets (called firewall packets) to the UDP send port on the server, from which the server will send RTP packets.
The content of the firewall packet contains a digital signature for authentication purpose.

The RTSP server figures out the external IP and port of the client's RTP receive socket by performing address translation using the firewall packets.

If the external IP of the firewall packet is the same as the percieved client IP address in the RTSP connection, the server treats the client as one that uses the same IP endpoint to host RTP and RTSP modules. In this case, the server ackownledges the firewall packet with a firewall answer packet packet in the reverse direction, and the server then starts to send the RTP packet stream using the perceived external IP and port, thus penetrating the firewall. Since the client initiats UDP traffic, this scheme works over symetric NATs, which STUN (rfc3489) does not.

However, if the server determines that the client is dual-hosted, i.e., the external IP address of the client's RTP receive socket is different than that of the client's RTSP socket (as seen from outside of the firewall)
, the server, as a protection against potential DoS attack, sends a FW request as the address challenging request, to confirm

that it is authorized to send RTP packets to this new host
(i.e., the RTP host). The RTP host should reply
by sending a FW response packet, with proper signature.
Only after the server has
received an answer to its request with a valid signature, that it
will begin to send RTP packets to that host.

1. Applicability Statement

This protocol is an extension to RTSP [RTSP_NEW]. It only solves
firewall traversal problem for multimedia streaming applications
under the following conditions:

1. The RTSP connection is over TCP or TLS;
2. The RTP streams are carried over UDP;
3. The server is in the open, ie. it is not behind a firewall;
4. The loss rate of UDP packets is not excessively high.
   This protocol depends on UDP packets to perform firewall
   exchange. Although multiple attempts are made during firewall
   exchange, the number of attempts is limited and would break
   down over high loss channel if all firewall packets get lost.

2. Requirements

An authenticated RTSP firewall solution should meet the following
requirements:

1. Minimal delay: a firewall traversal algirthm should not incur
   excessive delay on top of the normal RTSP delay.

2. Minimal price: for a client that is in the open, it should
   pay very little price (or no price at all)
   in terms of delay or complexity. That is to say, the firewall
   solution should quickly realize that there is no firewall
   in the middle and allow RTSP server to operate normally. Note
   the server still needs to challenge the client if it is
   dual-hosted in the open.

3. For a client that hosts RTP and RTSP modules from the same
   IP address, it should not perform additional message exchange
   for authentication.

Additionally, the authentication part of the RTSP firewall protocol
should meet the following requirements:

4. For the authentication feature to be used as a security
   measure against DoS attacks, it should be convenient
   and transparent enough that people actually turn them
   on. (See bullet 9, pp. 621 of [CRYP]).

5. Security should be designed as a basic feature, rather than
   a latter add-on. (See bullet 10,pp. 621 of [CRYP]).

3. Overview of Operation

The operation of the RTSP authenticated firewall tranversal involves
the following steps: 1) generation of a session key; 2) probing

of the firewall; 3) (optionally) Challenge of the dual-hosted receiver.

During step 1), if the RTSP server and client shares a secrete, S, then the server uses S to generate a 128-bit session key, K, and transform it into 128-bit RTSP session_ID:

    session_ID = E (K, S)

where E denotes the encryption transform AES (Advanced Encryption Standard, block cipher mode) [AES].

The RTSP server should ensure that the session_ID is unique in its current list of clients.

However, if the RTSP server and client has no pre-established shared secrete, K is used as session_ID, i.e.:

    session_ID = K

When client receives session_ID, it can obtain the session key, K to be used for all the streams in this RTSP session.

The server also records the perceived IP address of the client's RTSP socket, RTSP_ip(client).

Step 2) starts immediately after SETUP is complete on a media stream. We assume that in the SETUP response, the server includes a unique SSRC to be used as the sender SSRC in the corresponding RTP session. As a notation, FW(k) denotes the k-th firewall packet, where k=0, 1,2,...

In step 2), the client constructs a firewall packet FW(0) using the server SSRC, and signs it using K. It sends the resultant packet to the sender's IP address and port in the Transport header. To ensure delievery, the client sends multiple FW packets, FW(i), i=0,1,2,...9, with incremental sequence numbers and at 1 second interval. As soon as the client receives a FW response, it stops send FW packets.

When the server receives FW(r), it records the external IP, RTP_ip(client) of the client, and port, RTP_port(client) as seen in the IP header of FW(r).

It also records r to be used in the firewall response packegt.

From here on there are two possible paths for the protocol:

Path 1) If RTP_ip(client) == RTSP_ip(client), the client is a single hosted application, so the RTSP server responds with a series of FW response packet, FW_response(i), i=0,1,2,... which has the same format as FW packet, but 1) uses a different packet type
2) a different nonce that is 1 plus i plus the nonce in the request.
3) The sequence number in the response is r.
This response is signed using K, too.

Path 2) If RTP_ip(client) != RTSP_ip(client), the client
is deemed a dual hosted application. To prevent DoS attack,
the server should challenge RTP_ip(client) to verify its
willingness to accept RTP media stream. So the RTSP server
sends a succession of FW request packet, FW(c),c=0,1,2...
(each with proper digital signature as usual)
to port=RTP_port(client) on IP=RTP_ip(client), with new nounce,
and consequtive sequence numbers and 1 second interval.

On the client side, when it receives FW(c), where c is the
sequence number, if it is indeed willing to accept the
media stream, it will construct a FW response packet,
FW_response(r), r=0,1,2,..., in which:
1) the sequnece number is "c",
2) packet type is response type
3) nonce is r plus 1 plus nonce in FW(c).

This series of FW_response(r)'s are repeatedly
sent to the RTP send socket at the server, until
the client has received
RTP media packets, or that it has
reached maximum number of repeats (default is 10). The repeat
interval is 1 second by default. In each repetition the number "r" is
incremented by 1.

When the server receives the response to its FW request, it
is satisfied that RTP_ip(client) is willing to accept the
RTP media stream at port RTP_port(client).

Now and only now the server can start sending media packets
to port=RTP_port(client) on IP = RTP_ip(client).

If the server never recieves the response to its FW request,
event after max number of tries, it considers the client as a
attacker, hence will take immediate protective actions, such
as: terminating the RTSP connection, freeing any resources (
such as session ID and session key), and logging an attack
warning in its security log. Using the security log,
system administrator can trace back the attacker since
the attacker's identity might be carried in RTSP_ip(client)
and in the shared secrete.

4. Message Description

We now describe the format of the firewall packet.

4.1. Format of FW Pakcet

The firewall packet is a UDP
packet, consisting of a fixed length header and a body.

The header has 32 bits, which is divided in three subfields:
-- A 16-bit magic word, which is used to distinguish it
from a RTP or RTCP packet. The word has 16 zeros. Since the
first 16 bits of RTP packet contains the non-zero version
number, this magic work is gurranteed to be different

        than RTP/RTCP header.

     -- An 8-bit version number, currently set to 0. This version
        number is useful if the packet format needs to be upgraded
        in the future. A receiver that does not support this
        version number should ignore the entire FW packet.

     -- An 8-bit packet type, which indicates the purpose of the
        packet. The packet type is one of:
         0x00: request;
         0x01: response;

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |       MagicWord=0x0000         | version=00     |     type      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |               Server's RTP session  SSRC                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          sequence number   (w/ random offset in RTP-Info)       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          nunce (32 bits)                                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                 digital signature  (DS)                         |
 |                   .... (160 bits, SHA)                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The message body containes the following fields:

-- SSRC: this is the server's SSRC used for this RTP session. It
   can uniquely identify the RTP session at the server. Useful
   when one send port is used for multiple RTP clients by the
   server.

-- Randomized sequence number:
 This is the sequence number of the firewall
 exchange, randomized by the random offset signalled in the
 RTSP SETUP response of this stream.

 The legal sequence number, before randomization, is between 1 and 10
 -- clients should stop after 10 tries. This way if after subtracting
 random offset, the sequence number is not in the expected
 range, it is deemed malicious and dropped (with a warning
 log to alert the spoofing attempt).

-- nonce: a random 32-bit value

-- Digital signature:
 The default authentication scheme uses SHA-1 one way hash
 (see [SHA1]). Other hash algorithms can be adopted in the future,
 by either defining different packet types or new
 version numbers.

 The SHA-1 hash value has 160 bits, and is computed over the
 entire firewall packet.

6. Use cases

The following sub-sections illustrate use cases for RAFT, where RTSP is the singalling protocol in the multimedia stremaing application, while RTP/AVP (i.e., RTP over UDP) is the transport mechanism.

6.1 Client in the open

6.1.1 Client in the open and is single hosted

In this use case, the RAFT protocol exchange takes place in following steps:

1. After RTSP SETUP, the client starts firewall exchange by sending FW packets to server.

2. When server receives one of the FW packets, it learns that the client is in the open and is not dual-hosted, therefor it can start the RTP media stream any time after RTSP PLAY. It sends a FW response to the client as well. Note that even if the clients FW packets are all lost, the server still starts RTP stream, using the address and port in the client's SETUP request, after RTSP PLAY processing. So it is clear that no extra delay is introduced by RAFT.

6.1.2 Client in the open and is dual-hosted

In this use case, the RAFT protocol exchange takes place in following steps:

1. After RTSP SETUP, the client starts firewall exchange by sending FW packets to server.

2. When server receives one of the FW packets, it learns that the client is in the open but is dual-hosted, therefor it should challenge the RTP receiver before media stream can be sent. So instead of FW response, it sends a FW request to the client's RTP receiver host, in order to confirm that the host is willing to accept RTP media stream corresponding to the RTSP session.

3. When receiver host receives the FW request from the server, it answers it with an FW response.

4. When RTSP server receives the FW response, and is able to match the digital signature, it is ready to send RTP media stream anytime after RTSP PLAY.

In this scenario extra round-trip is required to perform receiver challenge using FW request/response mechanism. Such delay is unavoidable as long as authentication is needed to verify dual-hosted RTP receivers.

6.2 Client not in the open

6.2.1. Client not in the open and is single hosted

In this use case, the RAFT protocol exchange takes place in following steps:

1. After RTSP SETUP, the client starts firewall exchange by sending FW packets to server.

2. When server receives one of the FW packets, it learns that is not dual-hosted, and it also learns the external IP and port of the client.
   Now it can start the RTP
   media stream any time after RTSP PLAY.
   To inform the client so it will not repeat FW packets,
   the server sends a FW
   response to the client as well.
   However, if
   the clients FW packets are all lost, the server
   still starts RTP stream, using the address and port in
   the client's SETUP request, after RTSP PLAY processing.
   No extra delay is introduced by RAFT, but the client
   will not receive any RTP packets and would eventually
   timeout waiting for media data. Such a poor application
   behaviour is unavoidable due to excessive packet loss
   rate.

6.2.2. Client not in the open and is dual-hosted

In this use case, the RAFT protocol exchange takes place in following steps:

1. After RTSP SETUP, the client starts firewall exchange by sending FW packets to server.

2. When server receives one of the FW packets, it learns that the client is NOT in the open and appears to be dual-hosted, judging by external IPs of the RTSP address and RTP address of the client.
   Therefor it should challenge the RTP receiver before
   media stream can be sent. So instead of FW
   response, it sends a FW request to the client's RTP
   receiver host, in order to confirm that the host is
   willing to accept RTP media stream corresponding to
   the RTSP session.

3. When receiver host receives the FW request from the server, it answers it with an FW response.

4. When RTSP server receives the FW response, and is able to match the digital signature, it is ready to send RTP media stream anytime after RTSP PLAY.

7. Signalling For RAFT

7.1 Feature Tag

Following the RTSP extension methodology in [RTSP-CORE], we define a feature tag for RAFT:

rtsp.raft

7.2 Max Number of Attempts

We define a new parameter in SETUP response to signal, from server to client, the maximum number of attempts during RAFT packet exchange. By choosing this number to be zero, one can effectively turn off RAFT.

The new parameter is: max-raft-attempts

Example usage of the above singalling is shown below:

```
 C->S:  SETUP rtsp://foo.com/movie.mp4/trackID=1
        CSeq:  10
        Session: 0123456789012345
        Supported: rtsp.raft
        Transport: RTP/AVP;unicast;client_port=1234-1235;
                   max-raft-attempts=10

 S->C:  RTSP/1.0 200 OK
        CSeq: 10
        Session: 0123456789012345
        Supported: rtsp.raft
        Transport: RTP/AVP; unicast; server_port=5678-5679;
                   max-raft-attempts=10
```

8.Security Considerations

RAFT is an extension to the RTSP core protocol, with the explicit purposes to 1) solve firewall traversal problem;
2) solve the security problem in dual-hosted applications.

This section describes security considerations beyond those in [RTSP-CORE]

8.1 Attack Classifications

The main concern is the denaial or degrading of service attacks (DDOS attacks) that use a multimedia server to send RTP media streams to unwilling receivers, because such an attack provides substantial amplication given the generally large number of packets in multimedia streams.

There are two ways to launch DDOS attacks using a RTSP server equipped with RAFT:
- -- Denial or Degrading of service (DDOS) attacks can be launched by man-in-the-middle (MITM).
- -- DDOS attacks can be launched by a rogue client.

8.2 Man In The Middle Attacks

In this case, the attacker takes over control of one or more routers between RTSP server and client. It intercepts RAFT packets from client to server, modifies its IP header to make it look like that it comes form the target host. It also intercepts the challenge packets and fakes FW response to answer the challenges.

This type of attack requires:

1) The attacker has the abitlity to spoof IP header on the FW packets from client to server.

2). The attacker has the ability to intercept the RAFT challenges (which are FW request packets) and fake a FW response packet with correct digital signature.

To attack, the attacker should have the valid session key which only the RTSP client and the server should know.

8.3 Rogue Client Attack

Rather than simply intercepting traffic between RTSP client and server, the attacker may pretend to be a legitimate RTSP client, and tries to cause the server to send RTP streams to the victim host.

This type of attack requires:

1) The attacker is an authorized RTSP client.
2) The attacker still has to be able to diver to itself the RAFT challenge packets from server to the victim host, and fake responses to them.

8.4 Countermeasures

RAFT provides the following countermeasures:

1) MITM attacks can be prevented by the digital signature in each RAFT packet. We recommend that applications SHOULD not use RTSP session ID directly as session key, but rather establishes shared secrete which can be used to map session ID to the actual authentication session key. Such shared secrete effectively stops MITM attacks.

2) As to "Rogue Client Attacks", we recommend that the authentication key to be tied to host identity, whenever possible. This requires the availability of authentication server where the RTSP server can look up the proper key based on the identify of the intended RTP receiver host, rather than using the RTSP session key, because a rogue client does not have the authentication key of the victim host.

   If the service does exist for the RTSP server to look up authentication key based on RTP receiver host name or address, then the FW request that is used to challenge the RTP receiver, will use the authentication key associated with the RTP receiver host. And the FW response that is used to answer the aformentioned challenge, should also be signed with the RTP receiver's authenticaiont key. With such measures, rogue client attacks are no longer effective.

9. IAB Considerations

The IAB has studied the problem of "Unilateral Self Address Fixing", which is the general process by which a client attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism (RFC 3424 [17]). RAFT is an example of a protocol that performs this type of function. The IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements.

9.1 Problem Definition

From RFC 3424 [17], any UNSAF proposal should provide:

> Precise definition of a specific, limited-scope problem that is to be solved with the UNSAF proposal. A short term fix should not be generalized to solve other problems; this is why "short term fixes usually aren't".

The specific problems being solved by RAFT are:

o Provide a means for a RTSP client to obtain a binding on a NAT such that the open-Internet RTSP server's RTP packets can traverse the firewall
to reach the client.

RAFT does not address RTP over TCP.

9.2 Exit Strategy

From [17], any UNSAF proposal should provide:

> Description of an exit strategy/transition plan. The better short term fixes are the ones that will naturally see less and less use as the appropriate technology is deployed.

RAFT comes with its own built in exit strategy. As NATs and FWs are phased out through the deployment of IPv6, the RTSP server can signal to RTSP clients with a RAFT max-sequence number of 0, which will turn off RAFT.

9.3 Brittleness Introduced by RAFT

From [17], any UNSAF proposal should provide:

> Discussion of specific issues that may render systems more "brittle". For example, approaches that involve using data at multiple network layers create more dependencies, increase debugging challenges, and make it harder to transition.

RAFT introduces the following brittleness issue:

o RAFT assumes that the RTSP server exists on the public Internet. If the server is located in another private address realm, the client's FW packets
may or may not be able to reach the server.

o The binding obained by the client's

FW packet may be time limited. This binding life time
may need to be extended with periodical FW packets from
client to server. However, RAFT does not require client
to send
periodical FW packets. In our experience, the RTP stream
from server to client in itself is sufficient to maintain
the NAT binding on most, if not all, NATs.
However, the RTP traffic from server to client may not
enough to maintain the binding
if the NAT uses dynamic binding lifetimes to handle
overload, or if the NAT itself reboots during the streaming
process.

We note that RAFT works for ALL known NAT types, including
symetric ones.

9.4 Requirements for a Long Term Solution

From [17], any UNSAF proposal should provide:

Identify requirements for longer term, sound technical solutions
-- contribute to the process of finding the right longer term
solution.

Our experience with RAFT has led to the following requirements
for a long term solution to the NAT problem:

o To reliablly solve the RTSP NAT problem when both RTSP server
and client are in separate private address realms, at least
one RTP proxy should be provided in the public address realm.

10. IANA Considerations

A new feature tag should be registered with IANA:

-- feature tag: rtsp.raft

A new parameter in RTSP Transport header also needs to be
registered with IANA. This parameter is:
max-raft-attempts

Normative References

[RTSP_NEW] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M.,
"Real Time Streaming Protocol",
draft-ietf-mmusic-rfc2326bis-04.txt

[RTP_NEW] RFC3550 "Real-time Transport Protocol", July 2003

[STUN] RFC3489 "STUN - Simple Traversal of User Datagram Protocol
(UDP) Through Network Address Translators (NATs)",
March 2003

[SHA1] National Institute of Standards and Technology (NIST),
"Announcing the Secure Hash Standard", FIPS 180-1, U.S.
Department of Commerce, April 1995

[AES]   NIST, "Announcing the Advanced Encryption Standard",
FIPS 197, US. Department of Commerce, Nov. 2001

Informative References

[CRYP] Schneier, B., "Applied Cryptography - Protocols, Algrorithms,
and source code in C", 2nd edition, Wiley, 1996

Alcatel Reference No. 134188

ALCATEL

Alcatel Intellectual Property Department
Invention Disclosure Form (FIT) – US Version

Invention Title: **A Method of Extending a Streaming Server with Playlist Capabilities**

Inventor:**Thomas Zeng**

Full Name Thomas Zeng	Employee No. 140283	M/S	Phone 858 320 3125	Alcanet
Business Division MCG	Alcatel Company Packet V	Citizenship Canada	E-mail Address zeng@pvnetsolutions.com	
Supervisor Name, M/S, Phone No. Mike Seymour, 858 320-3119				
Home Address 10649 Indigo Way	City, State, Zip Code San Diego CA92127			County San Diego

1. What technical problem did you solve?

The problem is to add a flexible server-side playlist streaming capability with minimal impact on wireless terminals client.

2. What is the basic idea of your new solution? (Please make clear how your solution is different from the existing solutions)

Our solution piggybacks on existing session control protocols such as RTSP or SIP which are widely available in wireless terminals. In the case of RTSP, our solution involves an RTSP extension method, called PLAYLIST_PLAY. Central to our extension to basic session control protocol is the idea of 3 dimentional time. In existing protocols, the concept of time is naturally one dimentional, using so called "NPT time" or Normal Play Time. We have discovered that to support playlist seeking capabilities across playlist files and/or clips and/or NPT offsets within clips, the best approach is to define time as a triplet: <playlist_file_name, clip index, time offset>. This concept greatly simplied the algorithm to conduct playlist seek as required by end-user.

3. Please provide a short description of your new solution, including how it accomplishes what it does.

Our solution piggybacks on existing session control protocols such as RTSP or SIP which are widely available in wireless terminals. In the case of RTSP, our solution involves an RTSP extension method, called PLAYLIST_PLAY. Central to our extension to basic session control protocol is the idea of 3 dimentional time. In existing protocols, the concept of time is naturally one dimentional, using so called "NPT time" or Normal Play Time. We have discovered that to support playlist seeking capabilities across playlist files and/or clips and/or NPT offsets within clips, the best approach is to define time as a triplet: <playlist_file_name, clip index, time offset>. This concept greatly simplied the algorithm to conduct playlist seek as required by end-user.

For more information please refer to two references.

Two documents are attached which use RTSP as the context to add playlist capability, but that is not to say that our solution depends on RTSP. In fact, our playlist solution works also with other session control protocols such as PVConnect (a propriateary protocol developed by PacketVideo Network Solutions):

1.A document entitled: ServerSide Playlist System Design Document.

2. A email capturing the ideas. The email has been forwarded to Mr. Cordeiro and is entitled: Implementation of Seeking Across Playlists: Using "SET_PARAMETER" vs "PLAYLIST_PLAY".

6. What are the steps from when your new solution starts to when it stops (packet path, etc.)?

The invention starts when the wireless or wireline streaming client wishes to switch to the packets from a different clip or a diffent playlist file, it stops when the switch action is completed.

7. How may your invention be detected so that we can determine if someone is using your invention?

We can monitor the session control conversations between their client and server, and see if the following is happening:
That they (the server and the client) are using a three dimensional time to switch between clips and playlists.

8. Advantages of your new solution as compared to existing ones. Quantify if possible.

Some advantages are listed below:

1. Nested playlist file composition: one can easily then create playlist file to recursively include other playlist files. That is, a clip in a playlist can itself be composed of other clips. This is a powerful concept that opens doors for many application flexibilities.
2. Error handling is much easier than old solutions, hence it is desirable to others.
3. In the basic playlist mode, there is no impact to streaming client, and even in the full mode that allows random seeking across playlists and clips, the impact to streaming client is small, while most changes concentrated on server.

How to Enable Real-Time Streaming Protocol (RTSP) traverse Network Address Translators (NAT) and interact with Firewalls.
<draft-ietf-mmusic-rtsp-nat-02.txt>

Status of this memo

Abstract

This document describes six different types of NAT traversal techniques that can be used by RTSP. For each technique a description on how it shall be used, what security implications it has and other deployment considerations are given. Further a description on how RTSP relates to firewalls is given.

1. Definitions

1.1. Glossary

ALG	- Application Level Gateway, an entity that can be embedded in a NAT to perform the application layer functions required for a particular protocol to traverse the NAT [6]
ICE	- Interactive Connectivity Establishment, see [9].
DNS	- Domain Name Service
MID	- Media Identifier from Grouping of media lines in SDP, see [10].
NAT	- Network Address Translator, see [12].
NAT-PT	- Network Address Translator Protocol Translator, see [13]
RTP	- Real-time Transport Protocol, see [5].
RTSP	- Real-Time Streaming Protocol, see [1] and [7].
SDP	- Session Description Protocol, see [2].
SSRC	- Synchronization source in RTP, see [5].
TBD	- To Be Decided

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [4].

2. Introduction

Today there is a proliferative deployment of different flavors of Network Address Translator (NAT) boxes that in practice follow no open standards [12][18]. NATs cause discontinuity in address realms [18], therefore a protocol, such as RTSP, needs to try to make sure that it can deal with such discontinuities caused by NATs. The problem with RTSP is that, being a media control protocol that manages one or more media streams, it carries information about network addresses and ports inside itself. Because of this, even if RTSP itself, when carried over TCP for example, is not blocked by NATs, its media streams are often blocked by NATs when RTSP based streaming servers are deployed as is and without special provisions to support NAT traversal.

Like NATs, firewalls (FWs) are also middle boxes that need to be considered. They are deployed to prevent non-desired traffic/protocols to be able to get in or out of the protected network. RTSP is designed such that a firewall can be configured to let RTSP controlled media streams to go through with minimal implementation problems. However there is a need for more detailed information on how FWs should be configured to work with RTSP.

This document describes the usage of known NAT traversal mechanisms that can be used with RTSP. Following the guidelines spelled out in [18], we describe the required RTSP protocol extensions for each method, transition strategies, and we also discuss each method's security concerns.

This document is not based on RFC 2326 [1]. It is instead based and dependent on the updated RTSP specification [7], which is under development in IETF MMUSIC WG. The updated specification is a much-needed attempt to correct a number of shortcomings of RFC 2326. One important change is that the specification is split into several parts. So far only the updated core specification of RTSP is available in [7]. This document is one extension document to this core spec to document a special functionality that extends the RTSP protocol. This document is intended to be updated to stay consistent with the core protocol.

2.1. NATs

Today there exist a number of different NAT types and usage areas. The different NAT types are cited here from STUN [6]:

Full Cone: A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

Restricted Cone: A restricted cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

Port Restricted Cone: A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a packet, with source IP address X and source port P, to the internal host only if the internal host had previously sent a packet to IP address X and port P.

Symmetric: A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port. If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used. Furthermore, only the external host that receives a packet can send a UDP packet back to the internal host.

NATs are used on both small and large scale. The normal small-scale user is home user that has a NAT to allow multiple computers share the single IP address given by their Internet Service Provider (ISP). The large scale users are the ISP's themselves that give there users private addresses. This is done both for control and for lack of IP addresses.

Native Address Translation and Protocol Translation (NAT-PT) [13] is a mechanism used for IPv4 to IPv6 transition. This device is used to allow devices only having connectivity using one of the IP versions to communicate with the other address domain. If the other address domain is addressable through the use of domain names, then a DNS ALG assigns temporary IP addresses in the requestor's domain. The NAT-PT device translates this temporary address to the receivers true IP address and at the same time modify all necessary fields to be correct in the receiver's address domain.

2.2. Firewalls

A firewall (FW) is a security gateway that enforces certain access control policies between two network administrative domains: a private domain (intranet) and a pulic domain (public internet). Many organizations use firewalls to prevent privacy intrusions and malicious attacks to corporate computing resources in the private intranet [19].
A comparison between NAT and FW are given below:

1. FW must be a gateway between two network administrative domains, while NAT does not have to sit between two domains. In fact, in many corporations there are many NAT boxes within the intranet, in which case the NAT boxes sit between subnets.
2. NAT does not in itself provide security, although some access control policies can be implemented using address translation schemes.
3. NAT and FWs are similar in that they can both be configured to allow multiple network hosts to share a single public IP address. In other words, a host behind a NAT or FW can have a private IP address and a public one, so for NAT and FW there is the issue of address mapping which is important in order for RTSP protocol to work properly when there are NATs and FWs between the RTSP server and its clients.

In the rest of this memo we use the phrase "NAT traversal" interchangeably with "NAT/FW traversal" and "NAT/Firewall traversal".

3. Requirements

This section considers the set of requirements when designing or evaluating RTSP NAT solutions.

RTSP is a client/server protocol, and as such the targeted applications in general deploy RTSP servers in the public address realm. However, there are use cases where the reverse is true: RTSP clients are connecting from public address realm to RTSP servers behind home NATs. This is the case for instance when home surveillance cameras running as RTSP servers intend to stream video to cell phone users in the public address realm through a home NAT.

The first priority should be to solve the RTSP NAT traversal problem for RTSP servers deployed in the open.

The list of feature requirements for RTSP NAT solutions are given below:

1. MUST work for all flavors of NATs, including symmetric NATs
2. MUST work for firewalls (subject to pertinent firewall administrative policies), including those with ALGs
3. SHOULD have minimal impact on clients in the open and not dual-hosted

        - For instance, no extra delay from RTSP connection till arrival of media
4. SHOULD be simple to use/implement/administer that people actually turn them on
    - Otherwise people will resort to TCP tunneling through NATs
    - Address discovery for NAT traversal should take place behind the scene, if possible
5. SHOULD authenticate dual-hosted client transport handler to prevent DDOS attacks

4. Detecting the loss of NAT mappings

Several of the NAT traversal techniques in the next chapter use the fact that the NAT UDP mapping's external address and port can be discovered. This information is then utilized to direct the traffic intended for the local side's address to the external instead. However any such information is only valid while the mapping is intact. As the IAB's UNSAF document [18] points out, the mapping can either timeout or change its properties. It is therefore important for the NAT traversal solutions to handle the loss or change of NAT mappings, according to UNSAF.

First, it is important to ensure that there exists the possibility to send keep-alive traffic to minimize the probability of timeout. The difficulty is that the timeout timer can have varying length between different NATs. That is the reason why that UNSAF recommends usage of STUN to determine this timeout.

Secondly, it is possible to detect and recover from the situation where the mapping has been changed or removed. The possibility to detect a lost mapping is based on the fact that no traffic will arrive. Below we will give some recommendation on how to detect loss of NAT mappings when using RTP/RTCP under RTSP control.

For RTP session there is normally a need to have both RTP and RTCP functioning. The loss of a RTP mapping can only be detected when expected traffic does not arrive. If no data arrives after having issued a PLAY request and received the 200 response, one can normally expect to receive RTP packets within a few seconds. However, for a receiver to be certain to detect the case where no RTP traffic was delivered due to NAT trouble, one should monitor the RTCP Sender reports. The sender report carries a field telling how many packets the server has sent. If that has increased and no RTP packets has arrived for a few seconds it is very likely the RTP mapping has been removed.

The loss of mapping carrying RTCP is simpler to detect. As RTCP is normally sent periodically in each direction, even during the RTSP ready state, if RTCP packets are missing for several RTCP intervals, the mapping is likely to be lost. Note that if no RTCP packets are received by the RTSP server for a while, the RTSP server has the option to delete the corresponding SSRC and RTSP session ID, which means either the client could not get through a middle box NAT/FW, or that the client is mal-functioning.

5. NAT Traversal Techniques

There exist a number of NAT traversal techniques that can be used to allow RTSP to traverse NATs. However they have different features, they are applicable to different topologies; and the cost is also different. They also differ in their security considerations. In the following sections, each technique is outlined in details in terms of its advantages.

Not all of the techniques are yet described in the full details needed to actually use this document as a specification for how to use them. These sections are included to present comparison amongst the different methods in order for one to identify the most suitable method for a particular RTSP deployment scenario. There are methods that use protocols in early stage of standardization, such as TURN and ICE.

5.1. STUN

5.1.1. Introduction

STUN - "Simple Traversal of UDP Through Network Address Translators" [6] is a standardized protocol developed by the MIDCOM WG that allows a client to use secure means to discover the presence of a NAT between himself and the STUN server and the type of that NAT. The client then uses the STUN server to discover the address bindings assigned by the NAT. The protocol also allows discovery of the mappings timeout period and can be used in any keep-alive mechanism.

STUN is a client-server protocol. STUN client sends a request to a STUN server and the server returns a response. There are two types of STUN requests - Binding Requests, sent over UDP, and Shared Secret Requests, sent over TLS over TCP. We note here that for RTSP clients running on embedded devices, it may not be practical to require TLS be implemented on the embedded device (such as a cell phone). Therefore in the next section we propose to adapt RFC 3489 ([6]) so as to let RTSP use a subset of STUN packets/features for NAT traversal, but without requiring full implementation of STUN in an RTSP server or RTSP client. We note that RFC 3489 has provisions for STUN to be embedded in another application (see section 6 of [6]).

5.1.2. Using STUN to traverse NAT without server modifications

This section describes how a client can use STUN to traverse NATs to RTSP servers without requiring server modifications. However this method has limited applicability and requires the server to be available in the external/public address realm in regards to the client located behind a NAT(s).

Limitations:

- The server must be located in either a public address realm or the next hop external address realm in regards to the client.
- The client may only be located behind NATs that are of the full cone, address restricted, or port restricted type. Clients behind symmetric NATs cannot use this method.

Method:

A RTSP client using RTP transport over UDP can use STUN to traverse a full cone NAT(s) in the following way:

1. Use STUN to discover the type of NAT, if any, and the timeout period for any UDP mapping on the NAT. This is RECOMMENDED to be performed in the background as soon as IP connectivity is established. If this is performed prior to establishing a streaming session the possible delays in the session establishment will be reduced. If no NAT is detected, normal SETUP SHOULD be used.

2. The RTSP client determines the number of UDP ports needed by counting the number of needed media transport protocols sessions in the multi-media presentation. This information is available in the media description protocol, e.g. SDP. For example, each RTP session will in general require two UDP ports, one for RTP, and one for RTCP.

3. For each UDP port required, establish a mapping and discover the public/external IP address and port number with the help of the STUN server. If successful a mapping has been established:
   clients local address/port <-> public address/port.

4. Perform the RTSP SETUP for each media. In the transport header the following parameter SHOULD be included with the given values: "dest_addr" with the public/external IP address and port pair for both RTP and RTCP. To allow this to work servers MUST allow a client to setup the RTP stream on any port, not only even ports. The server SHOULD respond with a transport header containing an "src_addr" parameter with the RTP and

RTCP source IP address and port of the media stream.

5. To keep the mappings alive, the client SHOULD periodically send UDP traffic over all mappings needed for the session. STUN MAY be used to determine the timeout period of the NAT(s) UDP mappings. For the mapping carrying RTCP traffic the periodic RTCP traffic may be enough. For mappings carrying RTP traffic and for mappings carrying RTCP packets not frequent enough, keep alive messages SHOULD be sent. As keep alive messages, empty IP/UDP messages SHOULD be sent to the streaming servers discard port (port number 9).

If a UDP mapping is lost then the above discovery process is required to be performed again. The media stream needs to be SETUP again to change the transport parameters to the new ones. This will likely cause a glitch in media playback.

To allow UDP packets to arrive from the server to a client behind a restricted NAT, some UDP packets must first be sent to the server. The client, before sending a RTSP PLAY request, must send an empty or small UDP message, on each mapping, to the IP address given as the servers source address. To create minimum problems for the server these UDP packets SHOULD be sent to the server's discard port (port number 9) and contain no or very little data. To ensure that at least one UDP message passes the NAT, several messages are recommended to be sent.

For a port restricted NAT the client must send messages to the exact ports used by the server to send UDP packets before sending a RTSP PLAY request. This makes it possible to use the above described process with the following additional restrictions: For each port mapping, UDP packets needs to be sent first to the servers source address/port. To minimize potential effects on the server from these messages the following type of messages MUST be sent. RTP: An empty or less than 12 bytes large UDP message. RTCP: A correctly formed RTCP message.

The above described adaptations for restricted NATs will not work unless the server includes the "src_addr" "Transport" header parameter.

5.1.3. Embedding STUN in RTSP

This section outlines the adaptation and embedding of STUN within RTSP. This enables STUN to be used to traverse any type of NAT, including symmetric NATs. Any protocol changes are beyond the scope of this memo and is instead defined in TBD internet draft.

Limitations:

This NAT traversal solution (using STUN with RTSP) has limitations:

1. It does not work if both RTSP client and RTSP server are behind separate NATs.
2. The RTSP server may, for security reasons, refuse to send media streams to an IP different from the IP in the client RTSP requests. Therefore, if the client is behind a NAT that has multiple public addresses, and the client's RTSP port and UDP port are mapped to different IP addresses, RTSP SETUP will fail.

Deviations from STUN as defined in RFC 3489

Specifically, we differ from RFC3489 in two aspects:
1. We allow RTSP applications to have the option to perform "binding discovery" without authentication;
2. We require STUN server be co-located on RTSP server's media ports.

In order to allow binding discovery without authentication, the STUN server embedded in RTSP application would ignore authentication tag, and the STUN client embedded in RTSP application would use dummy authentication tag, as well.

In order to use STUN to solve NAT traversal when RTSP client is behind a symmetric NAT, STUN server must co-locate on RTSP server's media ports. This can be done, for instance, by embedding STUN server in RTSP server.

In fact, if STUN server is indeed co-located with RTSP server's media port, then a RTSP client using RTP transport over UDP can use STUN to traverse ALL types of NATs that have been defined in section 3.1. In the case of symmetric NAT, the party inside the NAT must initiate UDP traffic. The STUN Bind Request, being a UDP packet itself, can serve as the traffic initiating packet. Subsequently, both the STUN Binding Response packets and the RTP/RTCP packets can traverse the NAT, regardless of whether the RTSP server or the RTSP client is behind NAT.

Likewise, if a RTSP server is behind a NAT, then an embedded STUN server must co-locate on the RTSP client's RTCP port. In this case, we assume that the client has some means to establish TCP connection to the RTSP server behind NAT so as to exchange RTSP messages with the RTSP server.

To minimize delay, we require that the RTSP server supporting this option must inform its client the RTP and RTCP ports that the server intend to send RTP and RTCP packets, respectively.

To minimize the keep-alive traffic for address mapping, we also require that the RTSP end-point (server or client) sends and receives RTCP packets from the same port.

5.1.4. Discussion On Co-located STUN Server

In order to use STUN to traverse symmetric NATs the STUN server needs to be co-located with the streaming server media ports, i.e., the port from which RTP packets will be sent. This creates a de-multiplexing problem: we must be able to differentiate a STUN packet from a media packet. This will be done based on heuristics. This works fine between STUN and RTP or RTCP where the first byte has always present difference, but this can't be guaranteed to work with other media protocols.

5.1.5. ALG considerations

If a NAT supports RTSP ALG (Application Level Gateway) and is not aware of the STUN traversal option, service failure may happen, because a client discovers its public IP address and port numbers, and inserts them in its SETUP requests, when the RTSP ALG processes the SETUP request it may change the destination and port number, resulting in unpredictable behavior.

5.1.6. Deployment Considerations

For the non-embedded usage of STUN the following applies:

Advantages:

- Using STUN does not require RTSP server modifications; it only affects the client implementation.

Transition:

The usage of STUN can be phased out gradually as the first step of a STUN capable machine can be to check the presence of NATs for the presently used network connection. The removal of STUN capability in the client implementations will however most probably wait until no need at all exists to use STUN.

For the Embedded STUN method the following applies:

Advantages:

- STUN is a solution first used by SIP applications. As shown above, with little or no changes, RTSP application can re-use STUN as a NAT traversal solution, avoiding the pit-fall of solving a problem twice.
- STUN has built-in message authentication features, which makes it more secure. See next section for an in-depth security discussion.
- This solution works as long as there is only one RTSP end point in the private address realm, regardless of the NAT's type. There may even be multiple NATs (see figure 1 in [6]).
- Compares to other UDP based NAT traversal methods in this document, STUN requires little new protocol development (since STUN is already a IETF standard), and most likely less implementation effort, since open source STUN server and client have become available [21]. There is the need to embed STUN in RTSP server and client, which require a de-multiplexer between STUN packets and RTP/RTCP packets. There is also a need to register the proper feature tags.

Transition:

The usage of STUN can be phased out gradually as the first step of a STUN capable machine can be to check the presence of NATs for the presently used network connection. The removal of STUN capability in the client implementations will however most probably wait until there is no need at all to use STUN.

5.1.7. Security Considerations

To prevent RTSP server being used as Denial of Service (DoS) attack tools the RTSP Transport header parameter "destination" and "dest_addr" are generally not allowed to point to any IP address other than the one that RTSP message originates from. The RTSP server is only prepared to make an exception of this rule when the client is trusted (e.g., through the use of a secure authentication process, or through some secure method of challenging the destination to verify its willingness to accept the UDP traffic). Such restriction means that STUN does not work for NATs that would assign different IP addresses to different UDP flows on its public side. Therefore most multi-addressed NATs will not work with STUN.

In terms of security property, STUN combined with destination address restricted RTSP has the same security properties as the core RTSP. It is protected from being used as a DoS attack tool unless the attacker has ability to hijack RTSP stream.

Using STUN's support for message authentication and secure transport of RTSP messages, attackers cannot modify STUN responses or RTSP messages to change media destination. This protects against hijacking, however as a client can be the initiator of an attack, these mechanisms cannot securely prevent RTSP servers being used as DoS attack tools.

5.2. ICE

5.2.1. Introduction

ICE (Interactive Connectivity Establishment) [9] is a methodology for NAT traversal that is under development for SIP. The basic idea is to try, in a parallel fashion, all possible connection addresses that an end point may have. This allows the end-point to use the best available UDP "connection" (meaning two UDP end-points capable of reaching each other). The methodology has very nice properties in that basically all NAT topologies are possible to traverse.

Here is how ICE works. End point A collects all possible address that can be used, including local IP addresses, STUN derived addresses, TURN addresses. On each local port that any of these address and port pairs leads to, a STUN server is installed. This STUN server only accepts STUN requests using the correct authentication through the use of username and password.

End-point A then sends a request to establish connectivity with end-point B, which includes all possible ways to get the media through to A. Note that each of A's published address/port pairs has a STUN server co-located. B, before responding to A, uses a STUN client to try to reach all the address and port pairs specified by A. The destinations for which the STUN requests have successfully completed are then indicated. If bi-directional communication is intended the end-point B must then in its turn offer A all its reachable address and port pairs, which then are tested by A.

If B fails to get any STUN response from A, all hope is not lost. Certain NAT topologies require multiple tries from both ends before successful connectivity is accomplished. The STUN requests may also result in that more connectivity alternatives are discovered and conveyed in the STUN responses.

This chapter is not yet a full technical solution. It is mostly a feasibility study on how ICE could be applied to RTSP and what properties it would have. One nice thing about ICE for RTSP is that it does make it possible to deploy RTSP server behind NAT/FIRWALL, a desirable option to some RTSP applications.

5.2.2. Using ICE in RTSP

The usage of ICE for RTSP requires that both client and server be updated to include the ICE functionality. If both parties implement the necessary functionality the following step-by-step algorithm could be used to accomplish connectivity for the UDP traffic.

This assumes that it is possible to establish a TCP connection for the RTSP messages between the client and the server. This is not trivial in scenarios where the server is located behind a NAT, and may require some TCP ports been opened, or the deployment of proxies, etc.

Refer to [22] for the mapping of ICE to RTSP.

5.2.3. Implementation burden of ICE

The usage of ICE will require that a number of new protocols and new RTSP/SDP features be implemented. This makes ICE the solution that has the largest impact on client and server implementations amongst all the NAT/FW traversal methods in this document.

Some RTSP server implementation requirements are:
- Full STUN server features
- limited STUN client features
- Dynamic SDP generation with more parameters.
- RTSP error code for ICE extension

Some client implantation requirements are:
- Limited STUN server features
- Limited STUN client features
- RTSP error code and ICE extension

5.2.4. Deployment Considerations

Advantages:
- Solves NAT connectivity discovery for basically all cases as long as a TCP connection between them can be established in the first hand. This includes servers behind NATs. (Note that a proxy between address domains may be required to get TCP through).
- Improves defenses against DDOS attacks as media receiving client requires authentications, via STUN on its media reception ports. See [22] for more details.

5.3. Symmetric RTP

5.3.1. Introduction

Symmetric RTP is a NAT traversal solution that is based on requiring NATed clients to send UDP packets to the server's media send ports. In core RTSP, usage of RTP over UDP is uni-directional, where the server sends RTP packets to client's RTP port. Symmetric RTP is similar to connection-oriented traffic, where one side (e.g., the RTSP client) first "connects" by sending a RTP packet to the other side's RTP port, the recipient then replies to the originating IP and port.

Specifically, when the RTSP server receives the "connect" RTP packet from its client, it copies the source IP and Port number and uses them as delivery address for media packets. By having the server send media traffic back the same way as the client's packet are sent to the server, address mappings will be honored. Therefore this technique has the advantage of working for all types of NATs. However, it does require server modifications. Symmetric RTP is somewhat more vulnerable to hijacking attacks, which will be explained in more details in the section discussing security concerns.

5.3.2. Necessary RTSP extensions

To support symmetric RTP the RTSP signaling must be extended to allow the RTSP client to indicate that it will use symmetric RTP. The client also needs to be able to signal its RTP SSRC to the server in its SETUP request. The RTP SSRC is used to establish some basic level of security against hijacking attacks. Care must be taken in choosing client's RTP SSRC. First, it must be unique within all the RTP sessions belonging to the same RTSP session. Secondly, if the RTSP server is sending out media packets to multiple clients from the same send port, the RTP SSRC needs to be unique amongst those clients' RTP sessions. Recognizing that there is a potential that RTP SSRC collision may occur, the RTSP server must be able to signal to client that a collision has occurred and that it wants the client to use a different RTP SSRC carried in the SETUP response.

Details of the RTSP extension are beyond the scope of this draft and will be defined in a TBD RTSP extension draft.

5.3.3. Deployment Considerations

Advantages:

- Works for all types of NATs, including those using multiple IP addresses.
- Have no interaction problems with any RTSP ALG changing the client's information in the transport header.

5.3.4. Security Consideration

Symmetric RTP's major security issue is that RTP streams can be hijacked and directed towards any target that the attacker desires. The method has also no protection if client desires to initiate media streams to a target to launch DDOS attacks.

The most serious security problem is the deliberate attack with the use of a RTSP client and symmetric RTP. The attacker uses RTSP to setup a media session. Then it uses symmetric RTP with a spoofed source address of the intended target of the attack. There is no defense against this attack other than restricting the possible bind address to be the same as the RTSP connection arrived on. This prevents symmetric RTP to be used with multi-address NATs.

The hijack attack can be performed in various ways. The basic attack is based on the ability to read the RTSP signaling packets in order to learn the address and port the server will send from and also the SSRC the client will use. Having this information the attacker can send its own RTP packets containing the correct RTP SSRC to the correct address and port on the server. The destination of the packets is set as the source IP and port in these RTP packets.

Another variation of this attack is to modify the RTP binding packet being sent to the server by simply changing the source IP to the target one desires to attack.

One can protect oneself against the first attack by applying encryption to the RTSP signaling transport. However, the second variation is impossible to defend against. As a NAT re-writes the source IP and port this cannot be authenticated, which is required in order to protect against this type of DOS attack.

The random SSRC tag in the binding packet determines how well symmetric RTP can fend off streaming hijacking performed by parties that are not "men-in-the-middle".
This proposal uses the 32-bit RTP SSRC field to this effect. Therefore it is important that this field is derived with a non-predictive randomizer. It should not be possible by knowing the algorithm used and a couple of basic facts, to derive what random number a certain client will use.

An attacker not knowing the SSRC but aware of which port numbers that a server sends from can deploy a brute force attack on the server by testing a lot of different SSRCs until it finds a matching one. Therefore a server SHOULD implement functionality that blocks ports that receive multiple binding packets with different invalid SSRCs, especially when they are coming from the same IP/Port.

To improve the security against attackers the random tags length could be increased. To achieve a longer random tag while still using RTP and RTCP, it will be necessary to develop RTP and RTCP payload formats for carrying the random tag.

5.4. Application Level Gateways

5.4.1. Introduction

An Application Level Gateway (ALG) reads the application level messages and performs necessary changes to allow the protocol to work through the middle box. However this behavior has some problems in regards to RTSP:

1. It does not work when the RTSP protocol is used with end-to-end security. As the ALG can't inspect and change the application level messages the protocol will fail due to the middle box.

2. ALGs need to be updated if extensions to the protocol are added. Due to deployment issues with changing ALG's this may also break the end-to-end functionality of RTSP.

Due to the above reasons it is NOT RECOMMENDED to use an RTSP ALG in NATs. This is especially important for NAT's targeted to home users and small office environments, since it is very hard to upgrade NAT's deployed in home or SOHO (small office/home office) environment.

5.4.2. Guidelines On Writing ALGs for RTSP

In this section, we provide a step-by-step guideline on how one should go about writing an ALG to enable RTSP to traverse a NAT.

1. Detect any SETUP request.

2. Try to detect the usage of any of the NAT traversal methods that replace the address and port of the Transport header parameters "destination" or "dest_addr". If any of these methods are used, the ALG SHOULD NOT change the address. Ways to detect that these methods are used are:
   - For embedded STUN, watch for the feature tag "nat.stun". If any of those exists in the "supported", "proxy-require", or "require" headers of the RTSP exchange.
   - For non-embedded STUN and TURN based solutions: This can in some case be detected by inspecting the "destination" or "dest_addr" parameter. If it contains either one of the NAT's external IP addresses or a public IP address. However if multiple NATs are used this detection may fail.

   Otherwise continue to the next step.

3. Create UDP mappings (client given IP/port <-> external IP/port) where needed for all possible transport specification in the transport header of the request found in (1). Enter the public address and port(s) of these mappings in transport header. Mappings SHALL be created with consecutive public port number starting on an even number for RTP each stream. Mappings SHOULD also be given a long timeout period, at least 5 minutes.

4. When the SETUP response is received from the server the ALG MAY remove the unused UDP mappings, i.e. the ones not present in the transport header. The session ID SHOULD also be bound to the UDP mappings part of that session.

5. If SETUP response settles on RTP over TCP or RTP over RTSP as lower transport, do nothing: let TCP tunneling to take care of

NAT traversal. Otherwise go to next step.

6. The ALG SHOULD keep alive the UDP mappings belonging to the an RTSP session as long as: RTSP messages with the session's ID has been sent in the last timeout interval, or UDP messages are sent on any of the UDP mappings during the last timeout interval.

7. The ALG MAY remove a mapping as soon a TEARDOWN response has been received for that media stream.

5.4.3. Deployment Considerations

Advantage:

- No impact on either client or server
- Can work for any type of NATs

Transition:

An RTSP ALG will not be phased out in any automatically way. It must be removed, probably through the removal of the NAT it is associated with.

5.4.4. Security Considerations

An ALG will not work when deployment of end-to-end RTSP signaling security. Therefore deployment of ALG will result in that end-to-end security will not be used by clients located behind NATs.

5.5. TCP Tunneling

5.5.1. Introduction

Using a TCP connection that is established from the client to the server ensures that the server can send data to the client. The connection opened from the private domain ensures that the server can send data back to the client. To send data originally intended to be transported over UDP requires the TCP connection to support some type of framing of the RTP packets.

Using TCP also results in that the client has to accept that real-time performance may no longer be possible. TCP's problem of ensuring timely deliver was the reasons why RTP was developed. Problems that arise with TCP are: head-of-line blocking, delay introduced by retransmissions, highly varying congestion control.

5.5.2. Usage of TCP tunneling in RTSP

The RTSP core specification [7] supports interleaving of media data on the TCP connection that carries RTSP signaling. See section 10.13 in [7] for how to perform this type of TCP tunneling.

There is currently new work on one more way of transporting RTP over TCP in AVT and MMUSIC. For signaling and rules on how to establish the TCP connection in lieu of UDP, see [16]. Another draft describes how to frame RTP over the TCP connection is described in [17].

5.5.3. Deployment Considerations

Advantage:

- Works through all types of NATs where server is in the open.

Transition:

The tunneling over RTSP's TCP connection is not planned to be phased -out. It is intended to be a fallback mechanism and for usage when total media reliability is desired, even at the price of loss of real-time properties.

5.5.4. Security Considerations

The TCP tunneling of RTP has no known security problem besides those already present in RTSP. It is not possible to get any amplification effect that is desired for denial of service attacks due to TCP's flow control.

A possible security consideration, when session media data is interleaved with RTSP, would be the performance bottleneck when RTSP encryption is applied, since all session media data also needs to be encrypted.

5.6. TURN (Traversal Using Relay NAT)

5.6.1. Introduction

Traversal Using Relay NAT (TURN) [8] is a protocol for setting up traffic relays that allows clients behind NATs and firewalls to receive incoming traffic for both UDP and TCP. These relays are controlled and have limited resources. They need to be allocated before usage.

TURN allows a client to temporarily bind an address/port pair on the relay (TURN server) to its local source address/port pair, which is used to contact the TURN server. The TURN server will then forward packets between the two sides of the relay. To prevent DOS attacks on either recipient, the packets forwarded are restricted to the specific source address. On the client side it is restricted to the source setting up the mapping. On the external side this is limited to the source address/port pair of the first packet arriving on the binding. After the first packet has arrived the mapping is "locked down" to that address. Packets from any other source on this address will be discarded.

Using a TURN server makes it possible for a RTSP client to receive media streams from even an unmodified RTSP server. However the problem is that RTSP server may restrict that destinations other than the IP address that the RTSP message arrives from shall not be accepted. This means that TURN could only be used if the server knows and accepts that the IP belongs to a TURN server and the TURN server can't be targeted at an unknown address. Unfortunately TURN servers can be targeted at any host that has a public IP address by spoofing the source IP of TURN Allocation requests.

5.6.2. Usage of TURN with RTSP

To use a TURN server for NAT traversal, the following steps should be performed.

1. The RTSP client connects with RTSP server. The client retrieves the session description to determine the number of media streams.

2. The client establishes the necessary bindings on the TURN server. It must choose the local RTP and RTCP ports that it desires to receive media packets. TURN supports requesting bindings of even port numbers and continuous ranges.

3. The RTSP client uses the acquired address and port mappings in the RTSP SETUP request using the destination header. Note that the server is required to have a mechanism to verify that it is allowed to send media traffic to the given address. The server SHOULD include its RTP SSRC in the SETUP response.

4. Client requests that the Server starts playing. The server starts sending media packet to the given destination address and ports.

5. The first media packet to arrive at the TURN server on the external port causes "lock down"; then TURN server forwards the media packets to the RTSP client.

6. When media arrives at the client, the client should try to verify that the media packets are from the correct RTSP server, by matching the RTP SSRC of the packet. Source IP address of this packet will be that of the TURN server and can therefore not be used to verify that the correct source has caused lock down.

7. If the client notices that some other source has caused lock down on the TURN server, the client should create new bindings and change the session transport parameters to reflect the new bindings.

8. If the client pauses and media are not sent for about 75% of the mapping timeout the client should use TURN to refresh the bindings.

5.6.3. Deployment Considerations

Advantages:

- Does not require any server modifications.
- Works for any types of NAT as long as the server has public reachable IP address.

Transition:

TURN is not intended to be phase-out completely, see chapter 11.2 of [8]. However the usage of TURN could be reduced when the demand for having NAT traversal is reduced.

5.6.4. Security Considerations

An eavesdropper of RTSP messages between the RTSP client and RTSP server will be able to do a simple denial of service attack on the media streams by sending messages to the destination address and port present in the RTSP SETUP messages. If the attacker's message can reach the TURN server before the RTSP server's message, the lock down can be accomplished towards some other address. This will result in that the TURN server will drop all the media server's packets when they arrive. This can be accomplished with little risk for the attacker of being caught, as it can be performed with a spoofed source IP. The client may detect this attack when it receives the lock down packet sent by the attacker as being mal-formatted and not corresponding to the expected context. It will also notice the lack of incoming packets. See bullet 7 in section 5.6.2.

The TURN server can also become part of a denial of service attack towards any victim. To perform this attack the attacker must be able to eavesdrop on the packets from the TURN server towards a target for the DOS attack. The attacker uses the TURN server to setup a RTSP session with media flows going through the TURN server. The attacker is in fact creating TURN mappings towards a target by spoofing the source address of TURN requests. As the attacker will need the address of these mappings he must be able to eavesdrop or intercept the TURN responses going from the TURN server to the target. Having these addresses, he can set up a RTSP session and starts delivery of the media. The attacker must be able to create these mappings. The attacker in this case may be traced by the TURN username in the mapping requests.

The first attack can be made very hard by applying transport security for the RTSP messages, which will hide the TURN servers address and port numbers from any eavesdropper.

The second attack requires that the attacker have access to a user account on the TURN server to be able set up the TURN mappings. To prevent this attack the server shall verify that the target destination accept this media stream.

6. Firewalls

Firewalls exist for the purpose of protecting a network from traffic not desired by the firewall owner. Therefore it is a policy decision if a firewall will let RTSP and its media streams through or not. RTSP is designed to be firewall friendly in that it should be easy to design firewall policies to permit passage of RTSP traffic and its media streams.

The firewall will need to allow the media streams associated with a RTSP session pass through it. Therefore the firewall will need an ALG that reads RTSP SETUP and TEARDOWN messages. By reading the SETUP message the firewall can determine what type of transport and from where the media streams will use. Commonly there will be the need to open UDP ports for RTP/RTCP. By looking at the source and destination addresses and ports the opening in the firewall can be minimized to the least necessary. The opening in the firewall can be closed after a teardown message for that session or the session itself times out.

Simpler firewalls do allow a client to receive media as long as it has sent packets to the target. Depending on the security level this can have the same behavior as a full cone NAT or a Symmetric NAT. The only difference is that no address translation is done. To be able to use such a firewall a client would need to implement one of the above described NAT traversal methods that include sending packets to the server to open up the mappings.

7. Open Issues

Some open issues with this draft:

- At some point we need to recommend one RTSP NAT solution so as to ensure implementations can inter-operate. This decision will require that requirements, security and desired goals are evaluated against implementation cost and the probability to get the final solution deployed.
- The ALG recommendations need to be improved and clarified.
- The firewall RTSP ALG recommendations need to be written as they are different from the NAT ALG in some perspectives.

8. Security Consideration

In preceding sessions we have discussed security merits of each and every NAT/FW traversal methods for RTSP. In summary, the presence of NAT(s) is a security risk, as a client cannot perform source authentication of its IP address. This prevents the deployment of any future RTSP extensions providing security against hijacking of sessions by a man-in-the-middle.

Each of these has security implications.

Using STUN will provide the same level of security as RTSP with out transport level security and source authentications; as long as the server does not grant a client request to send media to different IP addresses.

Using symmetric RTP will have a slightly higher risk of session hijacking than normal RTSP. The reason is that there exists a probability that an attacker is able to guess the random tag that the client uses to prove its identity when creating the address bindings.

The usage of an RTSP ALG does not increase in itself the risk for session hijacking. However the deployment of ALGs as sole mechanism for RTSP NAT traversal will prevent deployment of encrypted end-to-end RTSP signaling.

The usage of TCP tunneling has no known security problems. However it might provide a bottleneck when it comes to end-to-end RTSP signaling security if TCP tunneling is used on a interleaved RTSP signaling connection.

The usage of TURN has high risk of denial of service attacks against a client. The TURN server can also be used as a redirect point in a DDOS attack unless the server has strict enough rules for who may create bindings.

9. IANA Consideration

This specification does not define any protocol extensions hence no IANA action is requested.

```
Thomas Zeng                Tel: 1-858-320-3125
PacketVideo
  Network Solutions        Email: zeng@pvnetsolutions.com
9605 Scranton Rd., Suite 400
San Diego, CA92121
```

10. References

10.1. Normative references

[1] H. Schulzrinne, et. al., "Real Time Streaming Protocol (RTSP)", IETF RFC 2326, April 1998.

[2] M. Handley, V. Jacobson, "Session Description Protocol (SDP)", IETF RFC 2327, April 1998.

[3] D. Crocker and P. Overell, "Augmented BNF for syntax specifica-tions: ABNF," RFC 2234, Internet Engineering Task Force, Nov. 1997.

[4] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[5] H. Schulzrinne, et. al., "RTP: A Transport Protocol for Real-Time Applications", IETF RFC 1889, January 1996.

[6] J. Rosenberg, et. Al., " STUN - Simple Traversal of UDP Through Network Address Translators", IETF RFC 3489, March 2003

[7] H. Schulzrinne, et. al., "Real Time Streaming Protocol (RTSP)", draft-ietf-mmusic-rfc2326bis-04.txt, IETF draft, June 2003, work in progress.

[8] J. Rosenberg, et. Al., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-01.txt, IETF draft, March 2003, work in progress.

[9] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for the Session Initiation Protocol (SIP)," draft-rosenberg-sipping-ice-00, IETF draft, February 2003, work in progress.

[10] G. Camarillo, et. al., "Grouping of Media Lines in the Session Description Protocol (SDP)," IETF RFC 3388, December 2002.

[11] G. Camarillo, J. Rosenberg, " The Alternative Semantics for the Session Description Protocol Grouping Framework," draft-camarillo-mmusic-alt-01.txt, IETF draft, June 2002, work in progress.

10.2. Informative References

[12] P. Srisuresh, K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Internet Engineering Task Force, January 2001.

[13] Tsirtsis, G. and Srisuresh, P., "Network Address Translation - Protocol Translation (NAT-PT)", RFC 2766, Internet Engineering Task Force, February 2000.

[14] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, Internet Engineering Task Force, December 1998.
[15] J. Postel, "internet protocol", RFC 791, Internet Engineering Task Force, September 1981.
[16] D. Yon, "Connection-Oriented Media Transport in SDP", IETF draft, draft-ietf-mmusic-sdp-comedia-04.txt, July 2002.
[17] John Lazzaro, "Framing RTP and RTCP Packets over Connection-Oriented Transport", IETF Draft, draft-lazzaro-avt-rtp-framing-contrans-00.txt, January 2003.
[18] D. Daigle, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, Internet Engineering Task Force, Nov. 2002
[19] R. Finlayason, "IP Multicast and Firewalls", RFC 2588, Internet Engineering Task Force, May 1999
[20] Krawczyk, H., Bellare, M., and Canetti, R.: "HMAC: Keyed-hashing for message authentication". IETF RFC 2104, February 1997
[21] Open Source STUN Server and Client, http://www.vovida.org/applications/downloads/stun/index.html
[22] Zeng, T.M.: "Mapping ICE (Interactive Connectivity Establishment) to RTSP", IETF draft, draft-zeng-mmusic-map-ice-rtsp-00.txt, Feb 2004

Note there is a typo below: it said:
.. whether this has a change to ever become the basis of a playlist
RFC.

I meant:

whether this has a chance to ever become the basis of a playlist RFC.

This Playlist Play method is not in RTSP standard (rfc2326), and could
be
considered an extension to that standard -- in fact, we are considering
submitting a proposal to IETF in the future.

```
>
> Near the end of the playlist SDD review  yesterday, we touched upon
the
> idea of using a new method, in lieu of "SET_PARAMETER", to request a
new
> playlist update or to seek to a new clip within the same playlist.
>
> Although we originally thought that we were too late, schedule-wise,
to
> make such a change, upon researching RTSP RFC more carefully,
thinking
> through it more, and talking to a few people after the meeting, I
have
> realized that, switching to a new method would most likely shorten
the
> overall developement cycle for playlist feature (i.e., 3.4.2
release).
>
> this release will affect the ease of error handling of the protocol
between the
> streamer and the playlist client, the flexiblity of such protocol to
adapt
> to future requirement changes (from Sony or from other customers),
and
> eventualy (perhaps less importantly), whether this has a change to
ever
> become the basis of a playlist RFC.
>
> 1. The two options
>    First, allow me to present the two options in front of us.
> *      SET_PARAMETER
>         This is the approach that we have taken probably since the
> beginning of SDD work. Dave has clearly documented it. Here is the
excerpt
> from page 9 of Dave's latest SDD (version 0.4):
>
>        ...
>        14.     Client application requests a new playlist after a
specified
> amount of time after providing user feedback.
>        15.     PMA server returns a new playlist file and URL.
```

72

```
>       16.     Client application calls a new SET_PARAM API on player
> engine and passes it the playlist URL to request streaming from the
new
> playlist.
>       17.     Player engine sends an RTSP SET_PARAMETER command to the
SM
> passing it the playlist URL.
>       18.     SM returns the NPT (normal play time) value at which the
> switch to streaming from the new playlist will occur.
>       19.     The player engine returns this NPT time value back up to
the
> player application so that it can update the display appropriately.
>
> An example SET_PARAMETER to update playlist and jump directly to clip
8,
> at end of current clip, is shown below:
>
>                       CLIENT REQUEST
>                       SET_PARAMETER rtsp://example.com/public/foo
RTSP/1.0
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 40
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_index:  8
>                       x-pv-activate: End-Of-Clip
>
>                       STREAMING MODULE RESPONSE
>                       RTSP/1.0 200 OK
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 24
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_index:  8
>                       x-pv-clip_ts: 65000
>
>
> *     PLAYLIST_PLAY
>
> In stead of using SET_PARAMETER in step 17 above, we follow the
spirit of
> RTSP extension mechanism (see section 1.5 of RFC2326), and use a new
> method, called PLAYLIST_PLAY, to support the new feature for seeking
> across playlists. Here is how it looks, in lieu of the message
exchange
> above:
>
>                       CLIENT REQUEST
>                       PLAYLIST_PLAY rtsp://example.com/public/foo
RTSP/1.0
>                       CSeq: 7
>                       Session: 1234567890
>                       Range:
```

```
> playlist_play_time=<rtsp:/public/foo/fool.ply, 8, 0>-; time=End-Of-
Clip
>
>                        STREAMING MODULE RESPONSE
>                        RTSP/1.0 200 OK
>                        CSeq: 7
>                        Session: 1234567890
>                        Range:
> playlist_play_time=<rtsp:/public/foo/fool.ply, 8, 0>-; npt=65000-
>
>
```

> The semantics of PLAYLIST_PLAY is almost the same as "PLAY" method
as
> defined in RFC2326, with the following exceptions:
> *   No RTP-Info header in the response (it is not necessary, since
the
> rtp time and seq are continuous -- we are effectively simulating live
> stream).
> *   Range type is new, specified in a new format designed
specifically
> for playlist support. This "playlist_play_time" is a tupple with the
url
> of the playlist file, the clip index, and NPT time inside the clip.
> *   The time parameter (see page 34 rfc2326) in Range header takes
on
> new reserved tokens:
> *   "NOW" means the time to execute the "PLAYLIST_PLAY" requst is
right
> now,
> *   "End-Of-Clip" means clip boundary
> *   "End-Of-Playlist" means end of current playlist
> *   In the response to PLAYLIST_PLAY, in the Range header, the
> "time=End-Of-Clip" is replaced with "npt=" parameter, to inform the
client
> that End-Of-Clip actually maps to NPT time of 65000. Meaning that the
> presentation of =<rtsp:/public/foo/fool.ply, 8, 0> will begin at
> NPT=650000. With this information, the client can update the clip
banner
> information accordingly. Client also has enough information to derive
the
> RTP timestamp corresponding to the first RTP packet for the requested
clip
> in the new playlist.
>
> Note the open ended range just means that the end time of the
playlist
> session is unknown, in line with Sony requirement.
>
>
> 2. Implementation analysis
>
>   In this section we try to compare the merits of the two
approaches.
> *   SET_PARAMETER approach:
>
>   To implement SET_PARAMETER option, the normal scenario is rather
> simple, but problem arises in two areas:

```
>       1.  Error handling is more complex.  The following example is
taken
> from section 3.2.4 of Dave's SDD:
>                       STREAMING MODULE ERROR RESPONSE
>                       RTSP/1.0 200 OK
>                       CSeq: 7
>                       Session: 1234567890
>                       Content-length: 24
>                       Content-type: text/parameters
>
>                       x-pv-playlist_url: /public/foo/foo1.ply
>                       x-pv-clip_ts: 64385
>                       x-pv-clip_index: 9
>                       x-pv-error_code: Requested clip not found
>
>           Notice that in an RTSP 200 OK response, we are actually
reporting
> an error !
>           In contrast, the same error can be reported, when request is
made
> by PLAYLIST_PLAY, using the 457 code in section 11.3.8 of RFC2326.
Simply:
>                          RTSP/1.0 457
>                          The Range specified a clip that is not found.
>
>           Note that 457 is already implement in Streamer, whilst the
> "x-pv-error_code" fields will have to be thought through and
implemented
> anew -- not a trivial task given that more than 33% of streamer
> development effort has been spent on error handling.
>
>           That is the fundamental reason why the new method approach
will
> end up saving us develeopment  time, we believe.
>
>       2.  SET_PARAMETER approach will be more difficult to adapt to
> potential new requirements. One such new potential feature is the
ability
> to random position to any synch point in any clip across playlists.
For
> Sony it is not possible since PMA client has timing only accurate to
> seconds.
>            In contrast, this new feature can be easily implemented by
> putting a non-zero value in the last field in the
"playlist_play_time"
> tupple. Done !
>
>
> 3. Closing thoughts
>
> Fundamentaly, because we are adding a new feature, i.e., controlling
a new
> type of resource -- playlist resource, it is more natural to extend
> RFC2326  via a method, rather than via some x-pv parameters for
> SET_PARAMETER method.
>
```

pv3 Server-Side Playlist System Design Document

76

Table of Contents

77

78

1 Introduction

This document provides the system design specification for pv3 MM system. It documents the changes required for the Bedrock Playlist release to meet the requirements of the Sony NetMusic project.

1.1 Purpose

The purpose of this document is to describe the system design changes and additions for the Bedrock Playlist release. Enough detail will be provided to enable the interface specification to be completed. The intended audience of this document is development engineers.

1.2 Scope

This document describes only the SDD changes relevant to support the server-side playlist feature.

1.3 Definitions, acronyms, and abbreviations

See Master Glossary in ClearCase

1.4 References

TBD

2 Overview of Server-Side Playlists

Server-side playlists provide the ability for the streaming server to combine streams from multiple sources (in sequence) and stream to a client in a single RTSP/RTP session. The client need not (and may not even) be aware that there are multiple media sources. This is useful for providing ad insertion capability, or for applications where uninterrupted streaming (from multiple sources) is desired - i.e. where the client doesn't have to explicitly request streaming from each new source.

PacketVideo's support for server-side playlists includes the ability to stream from a list of individual media sources. Initially this is restricted to VOD files only, but there is nothing inherently prohibiting this from being extended to live sources.

In addition, PacketVideo's support of server-side playlists includes the ability to update an active playlist from pvServer on the fly – during an active streaming session. The request to switch to a new playlist is on a per-session basis. So only the client requesting the switch will get the new playlist. When the server advances to the next clip in the playlist for that client, it will use the clips from the new playlist. Any and all other clients streaming from the playlist will continue streaming uninterrupted and totally unaware of anyone else streaming from the playlist.

Note that the manifestation of this feature was highly driven by the Sony project requirements (below) such that this solution may be more of a shorter-term solution. A more longer-term and more generic solution to this requirement may be provided in a future release.

2.1 Sony Net Music

The Sony Net Music project is the main driving force behind adding support for server-side playlists into pv3. Therefore, the following sections will detail the requirements of this service. *Note that the descriptions that follow illustrate one possible use case for the Server-Side Playlist feature. Much of functionality identified below is part of the Sony PMA implementation that Sony Networks will develop. Care must be observed in using this SDD material since the pv3 features that will be supported will be a small subset of the features identified here.*

2.1.1 Overview of Service

The Sony PMA service is an audio-only service whereby users are able to subscribe to listen to a custom "channel". Initially the user can select a channel according to simple categories such as Mood or Genre. Then through periodic user feedback, the user's preferences are updated and the channel becomes customized for each specific user. Each time the user provides more feedback; the channel becomes even more tailored to the user's preferences.

The list of songs for a specific channel is managed through the use of a playlist resident on the client application. The client application displays 3 songs at a time to the user with the current song being highlighted in the display. Each time the user advances to the next song; the displayed list gets updated in real time.

2.1.1.1 User Feedback

For each song the user has the ability to provide feedback to the Sony PMA server about whether they liked or disliked the song based on some predefined criteria. In case the user selects "liked" for a song, Sony's application sends this information back to the Personalization server to update users profile and (eventually) create an updated playlist. In case the user selects "disliked", the current song stops streaming and the application skips to the next song. Again in this case, Sony's player application sends this information back to the Personalization server to update users profile and (eventually) create an updated playlist.

2.1.1.2 Dynamic Playlists

In the case of the "liked" option above, the client application will update the playlist list after 5 songs from where the liked option was selected. However, in the case of the "disliked" option, the new playlist will be updated after 10 songs. In case user has streamed all the songs from playlist created by Personalization server, the application will get a new playlist (before the current playlist actually ends) and will continue streaming with a completely new playlist completely transparent to the user.

2.1.1.3 Billing

All user billing will be based on time. Thus there is a requirement to send information to the Sony server when the song has been played or has been interrupted with reason and how much time was streamed. This will allow them to track the number of songs left for the user when updating playlists. For example, if before starting the stream I had credit for 20 songs, and then I listen to 5 songs and then click dislike on 6th song; their PMA server will create the play list for 5 songs which will be updated on our server on 16th song of the initial playlist giving the user a sum total of 20 songs.

2.1.1.4 Authentication

User authentication will be done by the Sony PMA server when communicating with the client application.[1] While generating the playlist, the Sony PMA personalization server will create a unique Playlist ID for each playlist seen by the client and streaming server within the streaming session. *Note that we will need to pass on this Playlist ID as part of events enabling them to map the user information in their back end system.*

2.1.2 Detailed Service Features and Assumptions

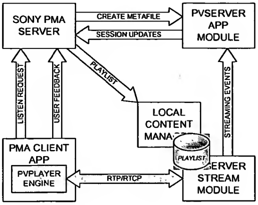
The following list provides more detailed information on the actual features of the service.

1. All content will be audio-only AAC encoded at the same bitrate
2. All content is pre-recorded content. No live content will be used.
3. All songs are categorized based on profiles by their personalization server using their pre-defined rules. This is performed outside of pvServer.
4. The break between songs during normal playback should be 2-3 seconds or less.
5. The user will have ability to repeat the same song again or skip back to the previous song or skip forward to the next (in case of "dislike").
6. Textual information will be displayed while the song is being played.
7. PMA server will do all authentication/authorization of users.
8. Dynamic playlist support is required including the capability of unlimited songs.
9. An individual playlist file will have approximately 30 songs.
10. Before creation of each playlist, the PMA server queries their back-end server to find out the amount of credit left in the user's account for streaming the songs. Depending upon time left, the PMA queue or buffer for songs will change. For example, a user selects a channel and PMA gets feedback from their back-end system that the user only has enough credit for streaming 20 songs. In this case the PMA queue of songs gets reduced to 20 songs as compared to 30-40 songs.
11. The billing will always be done on time basis for this service.
12. For each session when a playlist is created it will have unique Playlist ID created by PMA server which will have all relevant information about the user, their authentication etc.
13. PMA application will have an option where the user can check his time left on his credit, which needs to be updated in real time.
14. There is no time-based termination of the stream initiated by the server (a feature that is similar to the current "prepaidLive" feature in pvMM v.3.x).

2.1.3 Basic Architecture

The basic system architecture is as follows. The Sony PMA client application is built around the pvPlayer engine. The PMA client application connects to the PMA server to request content (playlist) and to provide user feedback. Streaming session establishment is through SDP files; therefore the PMA server must connect to the CreateMetafile service on the PacketVideo Application Module to request an SDP file. the client (pvPlayer engine) connects to the Streaming Module via RTSP to request an RTP streaming session. The Sony PMA server updates the playlists by pushing a new playlist file to the LCM. The following diagram illustrates the major components in this architecture and their various interconnections. For a more detailed look at the workflow of the components in this architecture, see Section 2.2.1 below.

[1] Note that additional authentication will be done by pv3 to prevent the SDP files from being shared among multiple users.

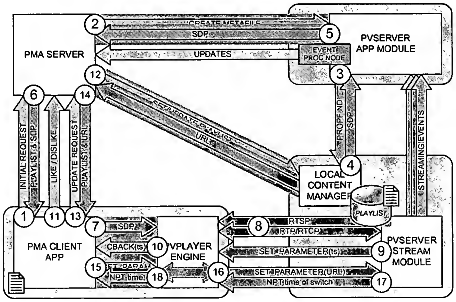


2.2 Server-Side Playlist Detailed Architecture Support

This section provides details on the impact of supporting server-side playlists on the various pv3 components.

2.2.1 Detailed Architecture Flow

The following diagram provides a detailed workflow of a server-side playlist session with the added requirements of the Sony service. The items in blue are changes required by PacketVideo components. Those is red are features that must be implemented by the system integrators and/or application developers. Items in gray are existing functionality within pv3.



Note that an initial assumption of this diagram is that all necessary media files are already resident on the LCM/SM.

1. PMA Server generates initial playlist(s) (possibly when user registers for the service). *Note that the initial playlist may be simply a default playlist that everyone starts out with that gets customized later.*
2. Client application makes request to PMA server to listen to a particular channel.

82

3. PMA server makes a request to the CreateMetafile service on the AM for an SDP file for a specific playlist.
4. CreateMetafile service makes the request to the LCM for the SDP file.
5. LCM opens the playlist file and appropriate media files, generates the SDP information, and returns it to the CreateMetafile service.
6. CreateMetafile service returns the SDP file to the requesting PMA server.
7. PMA server returns the playlist (generated previously) and the corresponding SDP file that was just retrieved to the client application.
8. Client application passes the SDP file to the player engine.
9. The player engine establishes a streaming session with the SM.
10. The SM sends periodic RTSP SET_PARAMETER messages to the player engine indicating a switch to a new clip within the playlist .
11. The player application calls back up to the client application passing it the timing information from the clip switch message from the SM.
12. Client application sends periodic updates of user preferences to PMA server based on user feedback.
13. PMA server creates a new playlist based on user preferences and pushes the new playlist out to LCM. The LCM returns the URL to be used when requesting streaming from this playlist.
14. Client application requests a new playlist after a specified amount of time after providing user feedback.
15. PMA server returns a new playlist file and URL.
16. Client application calls a new SET_PARAM API on player engine and passes it the playlist URL to request streaming from the new playlist.
17. Player engine sends an RTSP SET_PARAMETER command to the SM passing it the playlist URL.
18. SM returns the NPT (normal play time) value at which the switch to streaming from the new playlist will occur.
19. The player engine returns this NPT time value back up to the player application so that it can update the display appropriately.

Note that the addition of the SET_PARAMETER protocol between client and server in the above architecture is only there to support dynamic (on the fly) playlist updates in an uninterrupted streaming session and for (accurate) timing updates on clip transitions. For any customers that do not require these features, these features can be turned off as their architecture would not need to use this new client-server protocol. Any existing 3GPP-compliant player could be used for basic server-side playlist functionality.

For a different view on the overall system, see the call flow diagrams in the Appendix.

2.2.2 Components Affected

This section gives an overview of the components affected by the addition of server-side playlist support. Details on the individual components will be given in a later section.

2.2.2.1 Streaming Module

The pvServer Streaming Module needs several enhancements to support server-side playlists including:

- The SM must be able to recognize a playlist file by reading the beginning of the file.
- The SM must be able to open a playlist file and stream from multiple VOD files in succession within a single RTP session.
- The SM must be able to respond to a DESCRIBE request for a playlist file and generate the appropriate SDP.

83

- The SM must be able to send messages to the player engine when a switch to a new clip within a playlist occurs.
- The SM must be able to switch seamlessly between the VOD files such that the client sees an uninterrupted streaming session with no additional delays.
- The SM must be able to handle playlist update request from the player engine and send a response to the player engine when a playlist update occur.
- The SM must be able to handle repeat, skip forward, skip backward requests from the player engine.
- The SM must send new streaming events to the AM event service related to playlist and clip transitions.

For details on the SM changes, see Section 3.

2.2.2.2 Application Module Event Service

The pvServer Application Module needs to be configured to receive the new playlist streaming events generated by the SM as defined above. In addition, a new event-processing node may be written to process the information contained in the new events. *Note that the latter is outside the scope of the current Bedrock Playlist work.*

2.2.2.3 Local Content Manager

The pvServer LCM needs a new API to set and update the playlist files. In addition, the LCM must be able to open the playlist file and generate an SDP file that is relevant to the media clips contained in the playlist. Note that initially all of the media clips in the playlist must contain exactly the same codec types and configuration parameters. Therefore, the LCM need only open the first media file to be able to generate the SDP file. For more details on these changes, see Section 5.

2.2.2.4 pvPlayer Core Engine

The pvPlayer engine sees the media clips from the playlist as a continuous stream in a single RTP session. Therefore at a minimum, the existing pvPlayer engine could be used as is. However, to provide the functionality of random access within the playlist , updating the playlist on the fly, and receiving clip transition notifications, there are a few things that need to be added to the player engine including the ability to seek within a live stream and a new application APIs to allow the client application to pass information up to the streaming server and also to allow the streaming server to pass information back down to the client. For details on these changes, see Section 4.

3 Streaming Module Modifications

As described previously, the server-side playlist feature treats the streaming of all the clips defined in the playlist as one very long (possibly unending) session. All requests from the player to jump different parts of the playlist (i.e. to a different clip) will be handled through a new API. Skipping to different locations within a clip will NOT be supported through random positioning requests. Authentication of playlist requests will be treated similar to that of existing VOD and live requests.

3.1 General Playlist Handling

In order to support server-side playlists, the Streaming Module needs to do several things. During normal playback, the Streaming Module needs to seamlessly open each successive VOD source file in the playlist and continue streaming without interruption. This is seen by the player as a continuous RTP session.

Random positioning within clips will not be supported when streaming from a playlist file, including both pvAuthor and third-party generated content. Initially the Sony PMA server will use only pvAuthor-generated content. However, future deployments of this feature may use content authored by third-party encoders. However, currently pvServer does not support random positioning within third party content. So it was decided to not support random positioning altogether to make a consistent user experience. And since skipping to alternate clips is not implemented through random positioning, complete playlist functionality is still supported. *Note that this means that an independent PAUSE request is also not supported in addition to a PLAY request with a non-zero range header.*

3.1.1 Clip Skipping and Playlist Switching

When the SM receives a request to skip to a new clip/playlist, it will begin streaming from the requested clip/playlist without any interruption in the RTP streams. This may take effect immediately, at the end of the current clip, or at the end of the current playlist. *Note that a modified range header in the skip request governs the time at which this action takes place.*

These requests may contain a request to skip to a new clip in the current playlist, switch to a new playlist, or skip to a specific song in a new playlist (i.e. a combination of the first two). The behavior of the SM is dictated by the presence of various headers in the request. A playlist URL header indicates the URL from which to stream. This header is mandatory in all requests to ensure synchronicity between the server and client. The clip index header indicates the specific clip in the playlist that is desired. If this is missing, a default value of 1 is used so that the SM goes to the first song in the requested playlist. And lastly, a header is used to indicate the time at which this request should be satisfied. Possible values include immediately, at the end of the current clip, and at the end of the current playlist.

3.1.2 Precedence of Requests

If the SM receives successive requests that do not get fulfilled immediately (i.e. a playlist update), the last request received is the one that gets implemented. Any prior requests that have not been fulfilled will be overridden. The following are some example scenarios.

3.1.2.1 Playlist Update Overridden by Clip Skip

The SM is streaming clip C4 from playlist P2, then it receives playlist update request for P3 to take effect when C4 is done. But before C4 has completed the SM receives another request to skip back to clip C3 in playlist P1 immediately. Since the former request had not yet been satisfied, the latter request overrides the previous playlist update request (that would only take effect once C4 is done streaming). Thus, the SM immediately switches to playlist P1 and starts streaming clip C3.

3.1.2.2 Playlist Update Overridden by another Playlist Update

The SM is streaming clip C4 from playlist P2, then it receives playlist update request for P3. But before C4 has completed, the SM receives a request to update to playlist P4.

85

Both of these requests are to take effect at the end of the current clip. Since the former request to update to P3 had not yet been satisfied, when clip C4 completes, the SM will start streaming from the first clip in playlist P4. The request to update to playlist P3 was overridden.

3.2 Playlist RTSP Handling

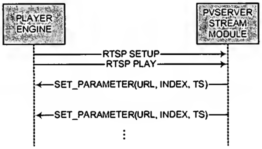
The Streaming Module must extend its RTSP support to handle a DESCRIBE request for a playlist file, to handle sending SET_PARAMETER commands to the client to signal a transition to a new clip in the playlist, and to receive a new PLAYLIST_PLAY command from the client requesting to switch to a new clip or to a new playlist altogether.

3.2.1 DESCRIBE Request Handling

When receiving a DESCRIBE request for a playlist URL, the SM must be able to respond with the appropriate SDP for the playlist content. Note that initially all of the media clips in the playlist must contain exactly the same codec types and configuration parameters. Therefore the SM need only open the first media file to be able to generate the appropriate SDP response. *Note, however, that this may not be true in the future if and when extended to allow codec changes, etc.* In addition, the SDP file for a playlist URL will have an open-ended session duration.

3.2.2 Signaling Clip Transitions

When the SM signals transitions from one clip to another within a playlist or to a new playlist altogether, the SM sends a SET_PARAMETER request to the client containing the URL of the (new) playlist, the index into the playlist of the next clip, and the NPT timestamp at which time the next clip will begin streaming. This is shown in green in the following diagram.



The following is example syntax of a normal transition request/response pair showing the passing the clip index and the NPT time at which point the new clip will begin.

STREAMING MODULE REQUEST

```
SET_PARAMETER rtsp://example.com/foo RTSP/1.0
CSeq: 6
Session: 1234567890
Content-length: 24
Content-type: text/parameters

playlist_play_time:</foo1.ply, 5, 0>
presentation_npt_time: 65000
```

86

CLIENT RESPONSE

```
RTSP/1.0 200 OK
CSeq: 6
Session: 1234567890
```

A note on the "playlist_play_time" parameter

Notice the structure of the value of the "playlist_play_time" attribute. This attribute is also used in a range header and signals the playlist and clip information needed to uniquely identify a clip. The syntax is as follows:

<URL, index, offset>

where

URL	is the playlist file URL,
index	is the (1-based) index into the playlist identifying the specific clip (e.g. clip 3), and
offset	is the time offset of the clip (i.e. start 5 seconds into the clip).

During a normal advance to the next clip, if the SM encounters an error while attempting to begin streaming from the next clip, the SM will skip the bad clip and streaming will then continue with the next good clip in the playlist. If the succeeding clip also contains an error, the SM will continue to skip to the next clip in the list until either the last clip is reached, the playlist is updated, or a successful clip is found. Only when a successful clip is reached will the SM send the SET_PARAMETER request.

In order to signal the error(s) to the client, the SET_PARAMETER request to the client indicating the new clip will include an additional parameter, "x-pv-error", indicating what happened (one parameter per error). This attribute will contain a list of every clip that encountered a problem. The following table illustrates the possible problems with the requested clip.

Symptom	Reason Code
Clip Missing	"Not found"
Clip Corrupt	"Corrupt"
Third-party content	"Non-PV Content"
Codec config change	"Codec Config Parameter"
Codec change	"Codec Change"
Number of media tracks different	"Media Track Number"

The following is example syntax of a transition request with clip errors.

STREAMING MODULE ERROR REQUEST

```
SET_PARAMETER rtsp://example.com/foo RTSP/1.0
CSeq: 7
Session: 1234567890
Content-length: 154
Content-type: text/parameters

playlist_play_time:</foo1.ply, 7, 0>
presentation_npt_time: 65000
x-pv-error: url=foo1.ply, index=5, reason="not found"
x-pv-error: url=foo1.ply, index=6, reason="codec change"
```

87

Note that for third-party clients, the SM will not send the periodic clip transition SET_PARAMETER messages. Third-party clients would not know to handle them, so we should not send them. Also, this should be a configurable option. We don't necessarily want the SM to always send these transition notifications for every deployment. This should have a configuration file entry to control the behavior for pvPlayer clients.

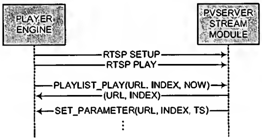
3.2.3 Random Positioning within a Clip or to Skip to a New Clip

When streaming from a playlist file, random positioning either within or outside the clip or playlist boundaries is not prohibited. *Note that an isolated PAUSE request is also not supported.* Random positioning within playlists will be supported in a future release.

3.2.4 Playlist Updates or Skipping to a Different Clip

When a client wants to switch to a new playlist or skip to a clip within a playlist, it sends a (proprietary) PLAYLIST_PLAY request to the SM containing information regarding which clip and/or playlist is requested. The request includes a new (proprietary) range header containing playlist URL, clip index, clip time offset (non-zero only if seeking into the clip), and the time at which this request is to be satisfied. Acceptable values for the time at which to satisfy the request include "now", "end of clip", and "end of playlist". A successful response to the request simply echoes the modified range header, minus the activation time. This is shown in blue in the following diagram.

The NPT timestamp at which time the new clip/playlist will begin playback is sent from the server to the client in a SET_PARAMETER call, as in normal clip transitions, that immediately follows the PLAYLIST_PLAY response. This is shown in green in the following diagram.



The following is example syntax of clip skip request/response pairs showing the details of passing the NPT time at which point the switch to the new clip will begin.

CLIENT PLAYLIST_PLAY REQUEST

```
PLAYLIST_PLAY rtsp://example.com/public/foo RTSP/1.0
CSeq: 7
Session: 1234567890
Range: playlist_play_time=</foo1.ply, 6, 0>-; time=now
```

STREAMING MODULE RESPONSE

```
RTSP/1.0 200 OK
CSeq: 7
```

88

```
Session: 1234567890
Range: playlist_play_time=</foo1.ply, 6, 0>-
```

STREAMING MODULE SET_PARAMETER REQUEST

```
SET_PARAMETER rtsp://example.com/public/foo RTSP/1.0
CSeq: 8
Session: 1234567890
Content-length: 24
Content-type: text/parameters

playlist_play_time:</foo1.ply, 6, 0>
presentation_npt_time: 65000
```

CLIENT RESPONSE

```
RTSP/1.0 200 OK
CSeq: 8
Session: 1234567890
```

If when the user makes a request to skip to a new clip/playlist, but the SM encounters an error while attempting to begin streaming from the clip/playlist, the SM will return an appropriate error code and will follow the behavior as defined in the following table. In addition, the response to the request may also include an additional parameter, "x-pv-error_code", indicating further what had happened.

Symptoms	Return Code	Behavior	x-pv-error_code
Everything okay	200 OK	Begins streaming from the specified clip/playlist	N/A
Playlist file not found	404 Not Found	Continues streaming from current playlist	N/A
Playlist file corrupt	500 Internal Server Error	Continues streaming from current playlist	"Playlist parse error"
Not authorized for secure playlist file	401 Unauthorized	Continues streaming from current playlist	"Secure playlist"
Public playlist w/ secure content	401 Unauthorized	Continues streaming from current playlist	"Secure content in public playlist"
Clip(s) not found	206 Partial Content	Begins streaming from next available (found) clip defined in the playlist. Indicated by clip index in response range header.	N/A Handled in the error code of the following S->C SET_PARAMETER request
Playlist found but all clips missing	404 Not Found	Continues streaming from current playlist	"All clips missing"
Clip corrupt	*Not verified during handling of PLAYLIST_PLAY request. Handled in the error code of the following S->C SET_PARAMETER request as described above.*		

When a playlist update is requested, the streamer will open the new playlist file, verify that it is valid, and prepare to switch to the playlist at the time specified in the request. If the new playlist itself is valid, but clips are missing, streaming will continue with the next available clip in the playlist after the one for which the skip request was made. In this case, the index value in the new range header in the PLAYLIST_PLAY response will indicate the next clip that will actually be streamed. If successive clips also contain errors, the SM will continue to skip to the next clip in the list until either the last clip is reached, the playlist is updated, or a successful clip is found. Only when a successful clip is reached will the SM send the SET_PARAMETER request. In other words, like in clip transitions, one skip request may account for more than one error.

CLIENT REQUEST

```
PLAYLIST_PLAY rtsp://example.com/public/foo RTSP/1.0
CSeq: 7
Session: 1234567890
Range: playlist_play_time=</foo1.ply, 5, 0>-; time=now
```

STREAMING MODULE ERROR RESPONSE

```
RTSP/1.0 206 Partial Content
CSeq: 7
Session: 1234567890
Range: playlist_play_time=</foo1.ply, 7, 0>-
```

If the playlist/clip switch actually takes place, the repsonse to the initial client request will be followed by a SET_PARAMETER request. But the SET_PARAMETER request itself may indicate errors if there are problems with the next clip(s). In this case, the errors indicated in the SET_PARAMETER request are the same as for the normal clip transitions described above in Section 3.2.2.

3.2.5 Maintaining Clip History on Playlist Update

The SM will not maintain any clip history when switching to a new playlist. When requested to switch to a new playlist file, since the request itself may contain both a request to switch to a new playlist as well as a request to skip to a specific clip within the playlist, there is no need for the SM to maintain any history as to past clips that were already streamed. If a client wants to go back to streaming from a past clip from a past playlist, it can generate the appropriate PLAYLSIT_PLAY request to do so that includes the proper playlist URL and clip index as described above in Section 3.2.4.

3.3 Authentication

The following sections discuss the process of user authentication during the initial client request to stream from a playlist file and also during the request to update a playlist.

3.3.1 Authentication of initial request

When an initial request to stream from a playlist is received, the SM will authenticate the requesting user if the appropriate authorization information is present in the URL query string. This is true regardless if the playlist file itself resides in a secure container or in a public container. In this case if the authorization succeeds, the client is authorized for all future playlist file updates in this session and may request a secure playlist file even if the initial request was for a public playlist.

If no authorization information is sent in the first request, the user is only able to request public playlists for the remainder of the session. Any request for a secure playlist will be denied.

For a secure playlist file, the content within the playlist file may contain public or secure content. However, for a public playlist file, all the content within the playlist must be public content (i.e. the SM has to check each clip and see if it's secure). If any of the clips within the public playlist reference secure content, the SM should reject the request, regardless if the user sent authorization information or not. The same goes for both the DESCRIBE and the first SETUP requests (until a valid session ID is generated).

3.3.2 Playlist update authentication

The request to update a playlist is governed by the security level established during the initial session establishment. Any authorization information received in a playlist update request will be ignored. If the user is requesting to switch to a secure playlist and has not already been authorized, the request will be denied even if the update request includes authorization information.

3.3.3 Digital signature calculation

When receiving a request for a secure playlist, the SM shall calculate the digital signature using the same fields in the URL query string as it does today. However, the URL over which the signature is calculated is not any specific MP4 file URL. It is the playlist URL. Note that this is not really a change. The digital signature is calculated over the request URL within which the query string and digital signature reside. In the normal VOD or live streaming case, this URL just happens to be the media URL. In the playlist case, this URL is the playlist file URL.

3.4 New Streaming Events

In order to support server-side playlists, a number of new streaming events need to be defined to signal the start of the playlist, the end of the playlist, and the start and end of each clip within a playlist. The following sections define these new events as well as give some example scenarios of when these events occur.

3.4.1 Event Definitions

Support for server-side playlists requires four new streaming events to be sent from the SM to the Application Module during a streaming session. See the IFS for details on the specific event XML syntax.

3.4.1.1 Playlist Start

The *Playlist Start* event signals that streaming has started from a new playlist (set of independent clips). This event contains a playlist ID, session ID (from the streaming session), number of clips, etc. When initial streaming from a playlist begins, the Playlist Start event follows the normal Setup Processed event. The Playlist Start event may also appear mid-session when a playlist has been updated and streaming begins from this updated playlist.

3.4.1.2 Playlist End

The *Playlist End* event signals that streaming has stopped from the current playlist. This event contains a playlist ID, session ID (from the streaming session), number of clips actually streamed, reason for ending, etc. When streaming from a playlist ends during normal playback, the Playlist End event precedes the normal Stream Terminated event. The Playlist End event may also appear mid-session when a playlist has been updated and streaming stops from the current playlist and a streaming will commence from an updated playlist.

3.4.1.3 Playlist Error

The *Playlist Error* event signals that streaming was intended to start from this playlist. However because of some error, streaming could not begin. For example the file may not be found or may be corrupt. This event contains a playlist ID, session ID (from the streaming session), and the reason for the error.

3.4.1.4 Clip Start

The *Clip Start* event signals that streaming has started from a new clip within a playlist. This event contains a clip URL, playlist ID, session ID (from the streaming session), etc.

3.4.1.5 Clip End

The *Clip End* event signals that streaming has stopped from the current clip within a playlist. This event contains a playlist ID, session ID (from the streaming session), reason for ending, amount of time (NPT) streamed, etc.

3.4.1.6 Clip Error

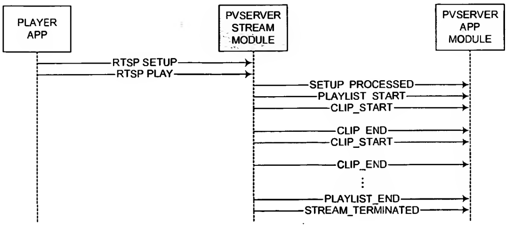
The *Clip Error* event signals that streaming was intended to start from this clip. However because of some error, streaming could not begin. For example the file may not be found or may be corrupt. This event contains a playlist ID, session ID (from the streaming session), clip ID, and the reason for the error.

3.4.2 Example Event Usage

The following sections illustrate how these new events behave in several streaming scenarios, including normal playback, skipping to the next song in a playlist, and updating a playlist mid-session.

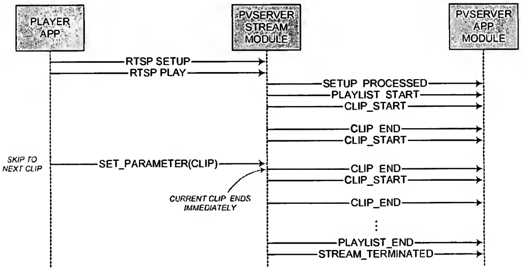
3.4.2.1 Normal Playback

During normal playback, a playlist file is opened and each clip is streamed in sequence. The *Playlist Start* event is sent to signal that streaming from a playlist is beginning. Then for each clip streamed from the playlist, a *Clip Start* and *Clip End* event is sent when the streaming from individual clip starts and ends, respectively. When the last clip in the playlist is done, the *Clip End* event is followed by the *Playlist End* event. The following diagram illustrates this process.



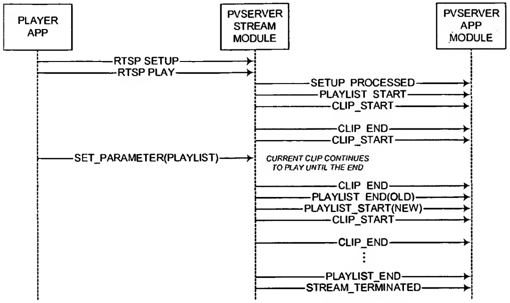
3.4.2.2 Skip Ahead (Random Position)

When skipping ahead to the next clip in a playlist, the *Clip End* event is sent signaling the end of the current clip, and the *Clip Start* event is sent signaling the beginning of the new clip in the playlist. Note that the Clip Start need not be for the next clip in normal playback order. This could be any clip from the current playlist. Again when the last clip in the playlist is done, the *Clip End* event is followed by the *Playlist End* event. The following diagram illustrates this process.

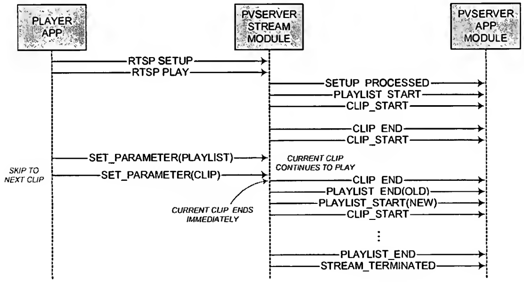


3.4.2.3 Update Playlist

When updating the playlist mid-session, an RTSP SET_PARAMETER method is used to signal the Streaming Module to use a new playlist file. When this happens, the current clip plays to the end. The SM sends a *Clip End* event followed by a *Playlist End* event to signal the end of playback from the current playlist. Then the SM sends a *Playlist Start* event to signal that streaming will begin from the new playlist. Again for each clip streamed from the new playlist, a *Clip Start* and *Clip End* event is sent when streaming from individual clip starts and ends, respectively. When the last clip in the playlist is done, the *Clip End* event is followed by the *Playlist End* event. The following diagram illustrates this process.



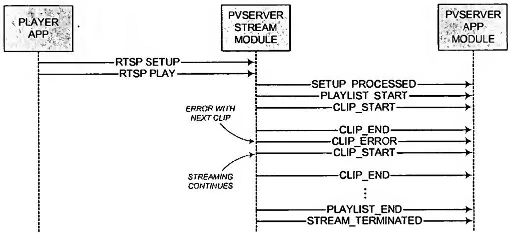
A possible variation to the above scenario occurs where just after the playlist update happens, the client signals to skip to the next song. In this case, the current clip terminates immediately after the skip request is received. The SM sends a *Clip End* event followed by a *Playlist End* event to signal the end of playback from the current playlist. Then the SM sends a *Playlist Start* event to signal that streaming will begin from the new playlist. And for each clip streamed from the new playlist, a *Clip Start* and *Clip End* event is sent when streaming from individual clip starts and ends, respectively. When the last clip in the playlist is done, the *Clip End* event is followed by the *Playlist End* event. The following diagram illustrates this variation.



94

3.4.2.4 *Clip Error*

During normal playback, a playlist file is opened and each clip is streamed in sequence. If there is an error while attempting to begin streaming from a clip, the SM sends a *Clip Error* event indicating the reason of the failure. Streaming will then continue with the next clip in the playlist.



Similar behavior occurs if the user makes an explicit request to skip to a clip, but the SM encounters an error while attempting to begin streaming from the clip. In this situation, the SM will again send a *Clip Error* event indicating the reason of the failure. Streaming will then continue with the next clip in the playlist after the one for which the skip request was made.

In either case above, if the succeeding clip also contains an error, the SM will send another Clip Error event and skip to the next clip in the list. If further errors are encountered, the SM will continue sending error events and skipping to the next clip until either the last clip is reached, the playlist is updated, or a successful clip is found.

4 pvPlayer Core Engine Changes

The following sections detail the changes to the pvPlayer Core Engine to support server-side playlists with the ability to dynamically update the playlist on the fly.

4.1 No Random Positioning for Playlist Streaming

The ability to random position within the stream does not directly affect server-side playlist behavior. The ability to skip to a different song within a playlist is implemented through a new API altogether. However, the impact of server-side playlists is that the streaming session has an undetermined end time and is always seen as a "live" stream by the pvPlayer engine. Thus random positioning is not allowed and the end user will not be able to seek within a single VOD file that is part of the playlist. This current behavior actually fits very well with the server behavior for playlists, as the SM will not allow random positioning when streaming from a playlist. The feature will be added in a future version of pvMM.

95

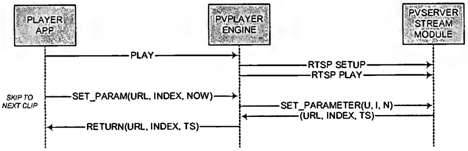
4.2 Pass-through Set Parameter API

In order to provide the ability to skip around within the playlist and also to provide live (on-the-fly) updates of the playlist, an application API is needed so that the player application can send a signal to the Streaming Module to tell it what to do. The signal to skip to another clip within the playlist was done this way since it was thought to be easier than making every skip request be implemented as a random position request. Plus it also saves an RTSP method. A random position request requires two RTSP commands (PAUSE followed by a PLAY) while using SET_PARAMETER only requires a single command. The signal to update the playlist was done in this manner so that the SM and the player will be in sync as to when the actual playlist file is switched.

When this API is called by the application, the player engine will send an RTSP SET_PARAMETER request to the streamer passing it the parameter(s) from the application. Any response values are sent back to the application in the return of the original API call or in a separate callback. Note that while this is a generic API, the only supported parameters from the application are a clip index, a new playlist URL, and the time at which to satisfy the request. In all cases, the SM will repond with the NPT time at which the switch to the next clip or to the new playlist will occur. This values get returned to the player application.

4.2.1 Skip to Next Clip

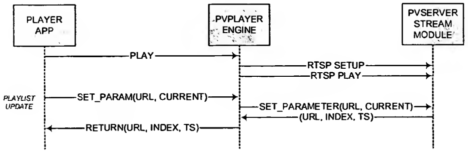
When requesting to skip to the next (or any other) clip in the playlist, the client application passes in the parameter names "x-pv-playlist_url", "x-pv-clip_index", "x-pv-activate" with the appropriate values. The parameters returned from the SM are the echoed URL and index (or new index if clip is different) and "x-pv-clip_ts" with the time value at which the switch to the next clip (or repeat of the current clip) will occur (and possibly an error code if any errors occur).



4.2.2 Playlist Update

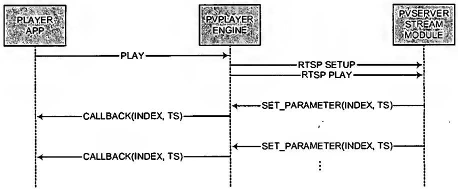
When requesting to skip to a new playlist, the client application passes in the parameter names "x-pv-playlist_url" and "x-pv-activate" with the appropriate values. Again, the parameters returned from the SM are the echoed URL, the clip index, and "x-pv-clip_ts" with the time value at which the switch to the new playlist will occur (and possibly an error code if any errors occur).

96



4.3 Receiving Set Parameter Method from Streamer

The SM signals transitions from one clip to another within a playlist by sending a SET_PARAMETER request to the client containing the NPT timestamp at which time the next clip will begin playback. The parameters returned from the SM are "x-pv-clip_index" indicating the index of the next clip to be streamed, and "x-pv-clip_ts" indicating the time value at which the switch to the next clip will occur (and possibly an error code if any errors occur). This is shown in green in the following diagram. Since the SM initiates the clip transition timing information, the player engine must make a callback into the player application to pass these values. This is shown in blue below.

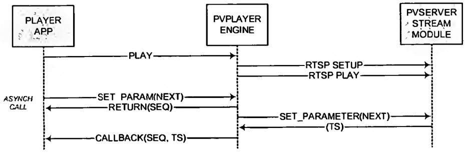


4.4 General API Issues

4.4.1 Asynchronous Callbacks

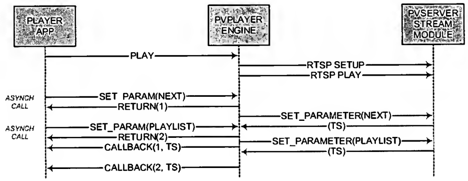
The API calls to skip to another clip and to update the playlist are shown above as synchronous calls where the response from the SM is received before the player engine responds to the calling application. This may not be the case depending on the implementation of the player engine. The API calls may have to be asynchronous so that the initial API call returns immediately so it doesn't block any further processing. The player engine then goes and sends the SET_PARAMETER to the SM. When the player engine gets the response from the SM, it must then call back into the player application to pass it the appropriate timing infomration as shown below.

97



However, in order to be able to manage posibly overlapping calls and responses, the return value to the calling application must include a sequence number that identifies the request so that when the callback is made, it can be associated back to the original request. And for callbacks that are not associated with any client request, as in a clip transition callback originated from SM, a null sequence number can be used.

The following example illustrates two requests, one for a clip skip and the other for a playlist update, that are made to the player engine. Since the callback from the first request may not have been received by the time the second request is made, there is the possibility that the callbacks could get mixed up if they didn't have the identifying sequence number.



4.4.2 Parameter Lists

Not only because it is directly needed to support the Sony project, it is beneficial in general to have the SET_PARAMETER APIs support passing a list of parameters in both directions. For example, a request to skip to the next clip passes the playlist URL and the clip index. And the response includes the clip index, the NPT timestamp, and possibly other information about the clip.

4.4.3 Registering for Callbacks

The player engine APIs should include a mechanism for the application to register for the specific parameters for which it wants a callback. This way, the application is only receiving callbacks that it knows it can handle. It also eliminates unnecessary processing by both the engine and the application. For a parameter(s) that has no registrants, the player engine will do nothing.

98

5 Infrastructure Changes

The following sections detail the changes to the various infrastructure components to support server-side playlists including the Local Content Manager and the Application Module Event Service.

5.1 Local Content Manager

The Local Content Manager is the remote content management "agent" controlling how content gets placed on the pvServer environment. Support for server-side playlists impacts two areas of the LCM, namely (playlist) file management and SDP file generation.

5.1.1 Playlist Management API

Just like other multimedia content files, the playlist files themselves need to be managed and made available to the pvServer Streaming Module. For playlist file management, the LCM will provide a new API to pass in the playlist. The LCM will take the playlist and save it to a file making it available to the SM and will then return to the caller the URL to be used for either retrieving the playlist directly from the LCM or when accessing this playlist in a streaming request. The LCM will be responsible for making sure there are no playlist filename conflicts, etc. when storing the playlist file. *Note that an alternate option is to allow the calling party to specify the exact URL at which to save the playlist file. In this case, there is no specific return value other than a success return code.*

The LCM can be configured to use Basic or Digest Authentication for ensuring that playlist files can be updated only by those parties who are authorized to do so. In addition, the LCM supports the secure transport of media content from a content provider to the pv3 Mobilemedia System through the use of Secure-HTTP.

5.1.2 CreateMetafile and SDP File Generation

In addition to basic content management, the LCM is responsible for generation of SDP files. To continue this function when server-side playlists are being used, the LCM must be able to read a playlist file when generating an SDP file. It must be able to open the first media file in the playlist to retrieve media attributes to place in the SDP file. Note however, that the SDP file for a playlist may have an open-ended session duration if dynamic playlists are being used.

5.2 Event Service

As mentioned above, the pvServer Application Module needs to be configured to receive the five new playlist streaming events generated by the SM as defined in Section 3.4. In addition, a new event-processing node may be written to process the information contained in the new events. *Note that the latter is outside the scope of the current Bedrock playlist work.*

Question: *Do we store the events in the database by default? Can we make this a configurable option on the AM?*

6 Playlist Syntax

A playlist file is a SMIL 2.0 compliant file, which is also a well-formed XML document. The playlist syntax only uses a subset of the actual SMIL syntax including only enough elements to define a list of media clips to be played in a sequential order. Extra features like switches, mutual exclusion containers, and priority classes are not supported. An example SMIL playlist file is given below.

```
<?xml version="1.0"?>
<smil>
    <clientData title="title" genre="genre" copyright="2003"/>
    <meta name="x-pv-playlist_id" value="123456789"/>
    <seq>
        <media src="rtsp:\\host\clips\clip1.mp4" dur="10s">
            <clientData title="title" author="author" artist="artist" album="album" genre="genre" copyright="2003"/>
        </media>
        <media src="rtsp:\\host\clips\clip2.mp4" dur="10s" />
        <media src="rtsp:\\host\clips\clip3.mp4" dur="10s" />
    </seq>
</smil>
```

For more details on the full playlist syntax, see the SDD and additionally http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmsrvsdk/htm/playlistreference.asp for Microsoft's use of SMIL-2.

7 Feature/Level0 Testing

7.1 Overview

This section defines the updates that are required to the Feature/Level0 testing of pvMM System specifically for the Bedrock release including server-side playlist support. Feature/Level0 testing targets testing of individual modules of the system at a black box level. Test harness code is developed where required to simulate valid/invalid inputs/responses of other modules in the system.

All Feature/Level0 tests should be part of the automated build tests wherever possible. Therefore the test cases should be implemented with this in mind.

Documentation must be provided with each test harness that is developed, which provides enough information for anyone to easily run the test cases provided. This should generally take the form of a man page, html description, and/or FAQ.

The following features should be tested at the feature/level0 level:

- SM Playlist File Handling
- SM playlist SDP generation
- New Streaming Events
- SM handling SET_PARAMETER requests/responses
- Player engine handling SET_PARAMETER requests/responses
- LCM SDP file generation
- Player engine APIs and callbacks
- Random Positioning with undefined range in SDP

7.2 Backwards Compatibility

Server-side playlist support doesn't necessarily require any additional support by pvPlayer. Therefore, an older (pre-Bedrock) version of pvPlayer should be able to work with pvServer to request streaming from a server-side playlist. The older players won't support dynamic playlists, or at least not the triggering of them by the client, however. But basic streaming from a playlist file should work just fine.

On the contrary to support for playlists by older players, an older server would reject a request to stream from a playlist file. This is true for both the Streaming Module as well as the LCM when generating SDP files. The LCM would return an error when requested to generate an SDP file for a playlist.

Appendix

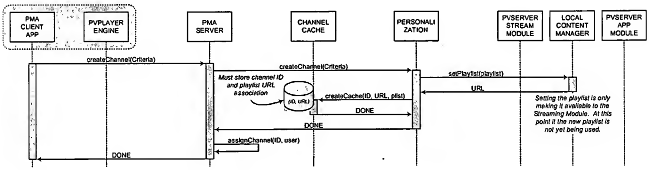
A Call Flow Diagrams

A.1 Normal User Scenarios

This section provides call flow walkthroughs of several normal user scenarios including channel setup and playlist creation, playlist updating, authentication, normal streaming playback, and clip skip requests.

A.1.1 Channel Setup and Initial Playlist Creation

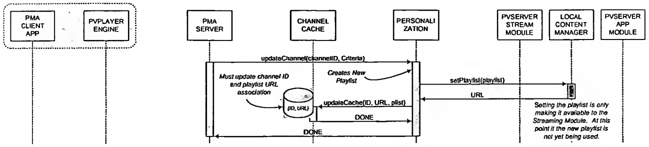
Most of this process is simply a reiteration of the original flow diagram from Sony. However as additional steps, when the channel is created, the Sony server (or some component thereof) needs to call an API on the pvServer Local Content Manager to pass it the playlist file for the channel that was just created. An association between channel ID and playlist URL must be stored so that when requested by a client to listen to a channel, the Sony server can request the appropriate SDP file with the playlist URL for that channel.



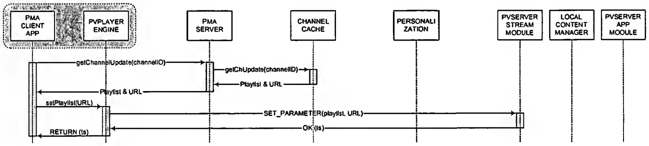
A.1.2 Playlist Update

A new playlist may be created in response to two events. First, if a user provides feedback to the PMA server regarding preferences to songs in the current playlist (i.e. "like" or "dislike"), the PMA server may create a new playlist taking into account these new user preferences thus further customizing the experience as seen by the end user. And second, when playback approaches the end of the current playlist, the PMA Server may generate a new playlist file so that the user can continue streaming uninterrupted from a new playlist when playback from the original playlist has completed. However, the actual point at which the PMA server generates a new playlist is totally governed by the PMA server itself.

When updating a playlist based on user preferences, the same basic process as channel creation is involved including calling the API on the Local Content Manager to pass it the playlist file that was just updated and updating the association between the channel ID and the playlist URL.

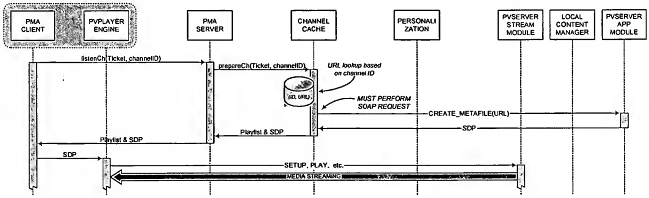

PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
updateChannel(channelID, Criteria)
Must update channel ID and playlist URL association
Creates New Playlist
setPlaylist(playlist)
URL
(ID, URL)
updateCache(ID, URL, plist)
DONE
DONE
Setting the playlist is only making it available to the Streaming Module. At this point it the new playlist is not yet being used.

In order to have the playlist update become active at the client, two additional things need to occur. First, the client application needs to contact the PMA Server to request an updated playlist. And second, the client application needs to call an API from the player engine to tell it to switch to a new playlist as shown below. More details on switching to a new playlist will be given in a later section.



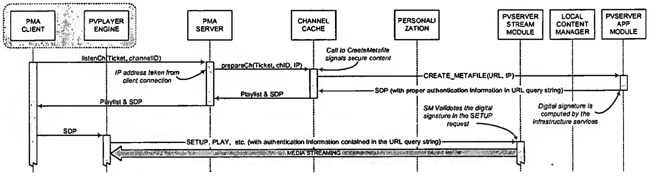
A.1.3 Basic Session Establishment

Session Establishment is done through SDP files. The Sony client application begins a session by making a request to the Sony PMA Server for a particular channel identified by a channel ID. This eventually results in a lookup based on channel ID to retrieve the playlist URL for making a SOAP call to the CreateMetafile service to request the appropriate SDP file. The SDP file is returned to the client application that in turn passes it to the pvPlayer engine. The player engine then establishes a streaming session with the streaming server using the information contained within the SDP file.



A.1.4 Authentication

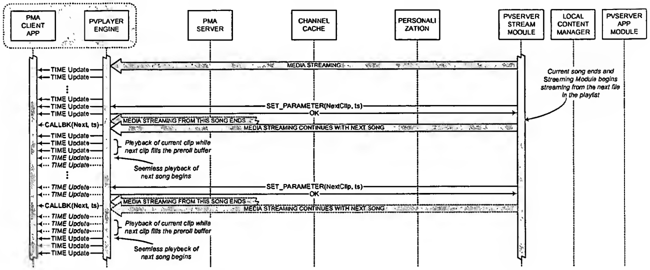
As discussed above, session establishment is done through the use of SDP files and the CreateMetafile Service on the PacketVideo Application Module. This fact has the side affect that an SDP file may be shared among clients; thus (possibly) allowing access to streaming content by someone that has not been authorized by the Sony PMA Server. Therefore to remedy this situation, an additional authorization step must be done using PacketVideo components. The use of the CreateMetafile Service for generating SDP files provides the Sony application the ability to use all of the existing PacketVideo authorization features, including IP address verification. When the Sony client application makes a request to the Sony PMA Server, a SOAP call to the pvServer Application Module CreateMetafile Service is made to request an SDP file. This call to the CreateMetafile service will indicate that this is for secure content. The CreateMetafile service will generate the SDP file that includes security information (including a unique digital signature with an appropriate expiration date) to be verified when the client connects to the pvServer Streaming Module. The Sony PMA Server will return the SDP file to the requesting application. The Sony client application will pass the SDP file to the player engine and the engine will begin the streaming session with the Streaming Module. When the player engine makes the first RTSP SETUP request to the Streaming Module, the URL in the request will contain the security information from the SDP file. The Streaming Module will then verify this information, including the digital signature, before the streaming session commences.


PMA CLIENT
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
listenCh(Ticket, channelID)
prepareCh(Ticket, chID, IP)
Call to CreateMetafile signals secure content
IP address taken from client connection
CREATE_METAFILE(URL, IP)
SDP (with proper authentication information in URL query string)
Playlist & SDP
Playlist & SDP
SM Validates the digital signature in the SETUP request
Digital signature is computed by the infrastructure services
SDP
SETUP, PLAY, etc. (with authentication information contained in the URL query string)
MEDIA STREAMING

A.1.5 Basic Streaming

The following diagram illustrates the communication between the pvServer Streaming Module and the pvPlayer engine during normal streaming playback, including advancing to the next song in the playlist when the current song is completed. When the streaming of one song is complete, the Streaming Module opens up the next file in the playlist and begins streaming the next file without any interruption to the client. The client sees a continuous stream of audio data. However before streaming from the next file begins, the Streaming Module sends a signal to the player engine indicating that streaming from a new clip will begin at a specific time (in terms of normal playback time or "NPT"). This information is passed back up to the application to use when updating the display appropriately when the song switched. The player engine is providing continual playback time updates to the player application. Thus since the player application has all of the necessary timing information (current playback time and time at which the switch to the new song will occur), it can determine when the transition from one song to the next occurs. This is all shown in the following diagram.

During the streaming of a song, the pvPlayer engine passes periodic timing updates to the PMA client application (shown in the black arrows labeled "TIME Update"). When a song transition occurs, the Streaming Module sends a "SET_PARAMETER" message to the pvPlayer engine with a parameter name of "NextClip" and a value of the time "ts" at which the next clip will start. The pvPlayer engine then makes a callback to the PMA client application (shown in green labeled "CALLBK") passing this information up to the client application. When media from the next song begins to stream, the current song continues to playback from data that is stored in the pre-roll buffer. While the data for the current song is emptying from the pre-roll buffer, data from the next song is filling in the buffer. When the data from the current song is finished, the next song begins to playback seamlessly without any gaps in the rendered audio.

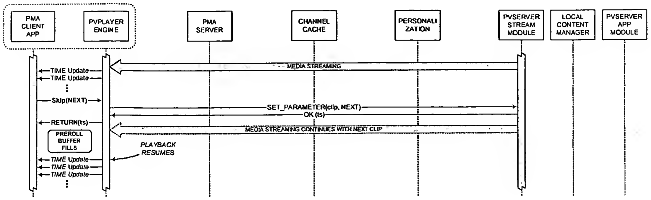

PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALI ZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
MEDIA STREAMING
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
SET_PARAMETER(NextClip, ts)
OK
Current song ends and Streaming Module begins streaming from the next file in the playlist
CALLBK(Next, ts)
MEDIA STREAMING FROM THIS SONG ENDS
MEDIA STREAMING CONTINUES WITH NEXT SONG
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
Playback of current clip while next clip fills the preroll buffer
Seemless playback of next song begins
TIME Update
TIME Update
SET_PARAMETER(NextClip, ts)
OK
CALLBK(Next, ts)
MEDIA STREAMING FROM THIS SONG ENDS
MEDIA STREAMING CONTINUES WITH NEXT SONG
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
Playback of current clip while next clip fills the preroll buffer
Seemless playback of next song begins

A.1.6 Advancing to the Next Song in the Playlist

The ability to skip ahead to the next song or back to the previous song is implemented through a new API provided to the PMA client application. This API allows the PMA Client to pass information about skipping through the playlist file up to the Streaming Module. Initially this API will only support skipping ahead one song in the playlist, one song back in the playlist, or repeating the current song.

When the player engine receives this API call, it relays the information to the Streaming Module. The response to the call to the Streaming Module is the NPT time at which the new song will start. This value is passed up to the PMA application to use in updating the display appropriately when the next song begins playing. The player engine is providing continual playback time updates to the player application. Thus the player application has all of the necessary timing information (current playback time and time at which the new song will begin playing) to determine when the transition from one song to another occurs.

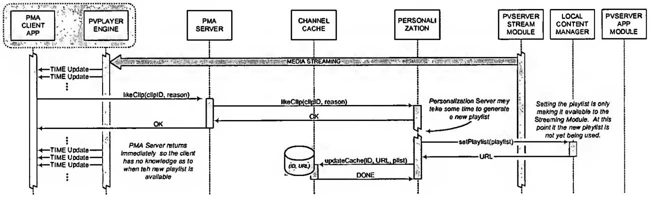
The following diagram illustrates this process for advancing to the next song in the playlist. This same process is followed for skipping back to the previous song in the playlist or repeating the current song.



A.1.7 User Feedback

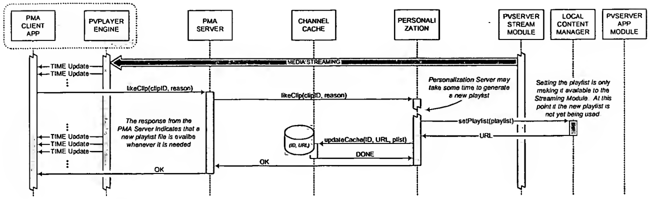
The process of user feedback is completely outside the scope of any PacketVideo components. If a user likes/dislikes a song, he/she interacts with the PMA client application accordingly. The PMA client application then communicates these preferences back to the PMA server to use for later playlist file updating.

The diagrams below illustrate two possible solutions. The first option below shows the feedback from the user being received by the PMA Server and the PMA Server sending a return code almost immediately (e.g. an asynchronous call). After the PMA Server acknowledges the user feedback, it proceeds to generate a new playlist (based on the user feedback) and make it available to the Streaming Module via a call to the pvServer Local Content Manager (LCM). This new playlist is now available for future use by the Streaming Module.


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
MEDIA STREAMING
TIME Update
TIME Update
likeClip(clipID, reason)
likeClip(clipID, reason)
Personalization Server may take some time to generate a new playlist
Setting the playlist is only making it available to the Streaming Module. At this point it the new playlist is not yet being used.
OK
OK
setPlaylist(playlist)
URL
TIME Update
TIME Update
TIME Update
PMA Server returns immediately so the client has no knowledge as to when teh new playlist is available
(ID, URL)
updateCache(ID, URL, pllst)
DONE

The second option shows the feedback from the user being received by the PMA Server and the PMA Server waiting to acknowledge the user feedback until the after new playlist is generated (e.g. a synchronous call). After the PMA Server receives the user feedback, it proceeds to generate a new playlist as before (based on the user feedback) and make it available to the Streaming Module via a call to the pvServer Local Content Manager (LCM). This new playlist is now available for future use by the Streaming Module. Once the playlist is generated, the PMA Server sends a return code back to the PMA Client. This allows the PMA Client to know exactly when the new playlist is available.

106


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALI ZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
MEDIA STREAMING
TIME Update
TIME Update
likeClip(clipID, reason)
likeClip(clipID, reason)
Personalization Server may take some time to generate a new playlist
Setting the playlist is only making it available to the Streaming Module. At this point it the new playlist is not yet being used
setPlaylist(playlist)
URL
The response from the PMA Server indicates that a new playlist file is evalibe whenever it is needed
(ID, URL)
updateCache(ID, URL, plist)
DONE
TIME Update
TIME Update
TIME Update
OK
OK

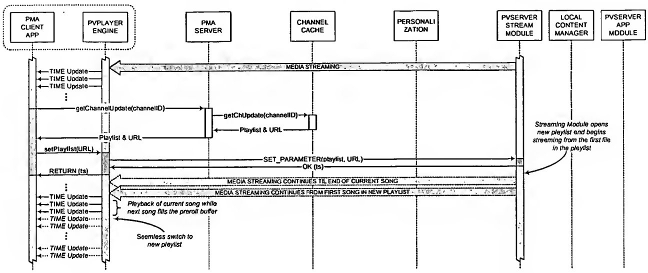
A.1.8 Playlist Switching

As mentioned previously, there are several events that may trigger the switch to a new playlist. If a user selects "like" for a particular song, a new playlist is generated using this new user preference information, and a switch to the new playlist will occur 5 songs later during normal playback. Similarly, if a user selects "dislike" for a particular song, a new playlist is generated using this new user preference information, and a switch to the new playlist will occur 10 songs later during normal playback. In both of these cases after user feedback is provided, playback from the current playlist continues while in parallel the PMA server generates a new playlist file. However in the case of "dislike", playback skips to the next song in the current playlist and playback resumes. In addition, one other event that may trigger the generation of a new playlist is the normal playback progress reaching the end of the current playlist. If continued playback is desired, a new playlist is generated and the switch to the new playlist will occur when playback of the last song of the current playlist has completed.

The PMA client application is always the one who initiates the switch to a new playlist. The PMA Client requests new playlist information from the PMA Server (as shown in the "getChannelUpdate()" call in the above diagram) and initiates the switch to the new playlist through the API call to the player engine (as shown in the "setPlaylist()" call in the diagram below). The PMA application knows when playback has reached 5 and/or 10 songs after user feedback was sent (as described above) and also when playback reaches the last song in the playlist. Therefore PMA client application is the one who is responsible for actually triggering the switch to the new playlist.

When the PMA Client calls the API in the pvPlayer engine, the engine sends a message to the Streaming Module telling it to switch to the playlist designated by the URL. When the Streaming Module receives this signal, it will continue streaming the current song until the end (or until a request to skip to the next clip is received), at which point it will immediately start streaming the first song defined in the new playlist without any delays (just like advancing normally to the next song within a playlist).

The response to the call to the Streaming Module is the NPT time at which the first song from the new playlist will start. Just like when skipping to the next song in the playlist, this time value is passed up to the PMA client application to use in updating the display appropriately when the next song begins playing. Because the player engine is providing continual playback time updates to the player application, the PMA client application has all of the necessary timing information (current playback time and time at which the next song will begin playing) to determine when the transition from the current song to the first song in the new playlist will occur.


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
MEDIA STREAMING
TIME Update
TIME Update
TIME Update
getChannelUpdate(channelID)
getChUpdate(channelID)
Playlist & URL
Playlist & URL
setPlaylist(URL)
SET_PARAMETER(playlist, URL)
OK (ts)
RETURN (ts)
Streaming Module opens new playlist and begins streaming from the first file in the playlist
MEDIA STREAMING CONTINUES TIL END OF CURRENT SONG
MEDIA STREAMING CONTINUES FROM FIRST SONG IN NEW PLAYLIST
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
Playback of current song while next song fills the preroll buffer
Seemless switch to new playlist
TIME Update
TIME Update

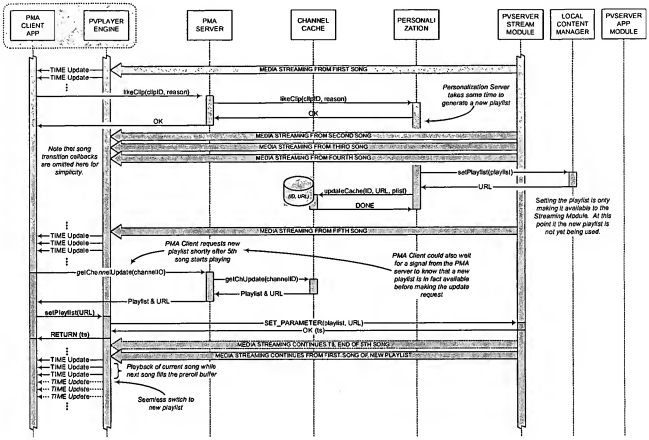
It should be noted that in all of these cases ("like", "dislike", and normal playback to the end of a playlist), the switch to the new playlist is always a seamless process as seen by the user. The end user will not have to suffer any delays or buffering of the streaming data. Playback continues without any interruption. Only when the user selects "dislike" will the user see a delay while the Streaming Module skips to the next song in the playlist (or to the first song in the new playlist) and while the player engine buffers the necessary data (i.e. fills the pre-roll buffer). But this is independent of the actual playlist update process.

A.1.9 Walkthrough

This section provides a walkthrough of the complete process of normal streaming, the user providing feedback, and switching to a new playlist. For clarity purposes, the normal playback signals of song switches are not shown. The diagram shows the user selecting "like" during the first song. This causes the PMA Server to generate a new playlist while streaming continues. *Note that the process of generating a new playlist may take some time as indicated in the diagram by several songs playing back before the new playlist file actually gets set on the LCM.*

After song 1 has completed, the Streaming Module continues to stream songs 2, 3, 4, and 5 while sending signals (not shown) at the start of each successive song. When song 5 begins, the PMA Client knows (because of the previous user feedback) that it must now request a new playlist file to be used after the current song has ended. The PMA Client makes the appropriate call to the pvPlayer engine, which in turn passes this information up to the Streaming Module. The response to the call to the Streaming Module is the NPT time at which the first song from the new playlist will start. This time value is passed up to the PMA application to use in updating the display appropriately when the first song from the new playlist begins playing.

The Streaming Module continues to stream song 5 until it completes normally at which point it begins streaming the first song defined in the new playlist. This is all done seamlessly so that the user sees no delays between song 5 and the first song in the new playlist.

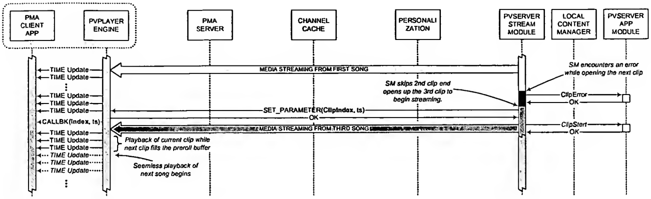

PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALI ZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
TIME Update
MEDIA STREAMING FROM FIRST SONG
likeClip(clipID, reason)
Personalization Server takes some time to generate a new playlist
OK
MEDIA STREAMING FROM SECOND SONG
MEDIA STREAMING FROM THIRD SONG
MEDIA STREAMING FROM FOURTH SONG
Note that song transition callbacks are omitted here for simplicity.
setPlaylist(playlist)
URL
updateCache(ID, URL, plist)
(ID, URL)
DONE
Setting the playlist is only making it available to the Streaming Module. At this point it the new playlist is not yet being used.
MEDIA STREAMING FROM FIFTH SONG
PMA Client requests new playlist shortly after 5th song starts playing
PMA Client could also wait for a signal from the PMA server to know that a new playlist is in fact available before making the update request
getChannelUpdate(channelID)
getChUpdate(channelID)
Playlist & URL
setPlaylist(URL)
SET_PARAMETER(playlist, URL)
RETURN (ts)
OK (ts)
MEDIA STREAMING CONTINUES TIL END OF 5TH SONG
MEDIA STREAMING CONTINUES FROM FIRST SONG OF NEW PLAYLIST
Playback of current song while next song fills the preroll buffer
Seamless switch to new playlist

A.2 Error Scenarios

This section provides call flow walkthroughs of several error scenarios including normal playback clip transition errors, clip skip request errors, and playlist update request errors.

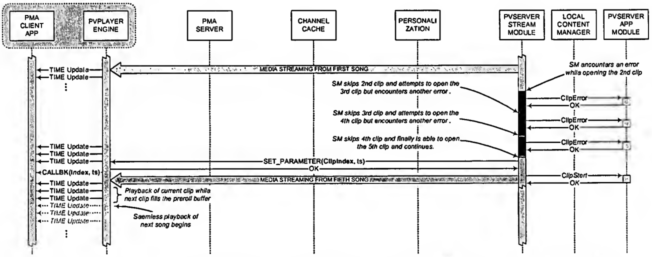
A.2.1 Single Error in Normal Clip Advance

This section provides a walkthrough of the behavior when the SM encounters a problem with the next clip in the playlist during normal clip advancing. The SM will skip over the problem clip and go to the next clip and try to continue streaming. The following call flow includes the streaming events sent to the AM to clarify the interaction of the event behavior along with the client RTSP behavior. Notice the addition of the clip index in the SET_PARAMETER call from the SM to the client. This signals that the normal next clip was skipped and streaming is continuing with the third clip in the playlist.


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
TIME Update
MEDIA STREAMING FROM FIRST SONG
SM encounters an error while opening the next clip
SM skips 2nd clip and opens up the 3rd clip to begin streaming.
ClipError
OK
SET_PARAMETER(ClipIndex, ts)
CALLBK(Index, ts)
MEDIA STREAMING FROM THIRD SONG
ClipStart
Playback of current clip while next clip fills the preroll buffer
Seamless playback of next song begins

A.2.2 Multiple Errors in Normal Clip Advance

This section provides a walkthrough of the behavior when the SM encounters problems with multiple clips in the playlist during normal clip advancing. The SM will skip over the problem clip and go to the next clip and try to continue streaming. If successive problems are encountered, the SM will keep skipping to the next clip until either a good clip is found, the end of the playlist is reached, or a request to switch to a new playlist is received. The following call flow includes the streaming events sent to the AM to clarify the interaction of the event behavior along with the client RTSP behavior. Note that the SM will send a streaming event to the AM for every attempted clip.

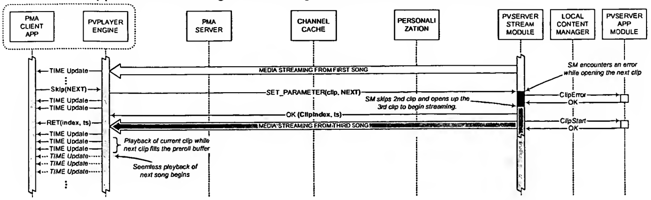

PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
TIME Update
MEDIA STREAMING FROM FIRST SONG
SM encounters an error while opening the 2nd clip
SM skips 2nd clip and attempts to open the 3rd clip but encounters another error.
SM skips 3rd clip and attempts to open the 4th clip but encounters another error.
SM skips 4th clip and finally is able to open the 5th clip and continues.
ClipError
OK
SET_PARAMETER(ClipIndex, ts)
CALLBK(Index, ts)
MEDIA STREAMING FROM FIFTH SONG
ClipStart
Playback of current clip while next clip fills the preroll buffer
Seamless playback of next song begins

Notice again the addition of the clip index in the SET_PARAMETER call from the SM to the client. This signals that the three clips were skipped and streaming is continuing with the fifth clip in the playlist.

A.2.3 Single Error in Skip Request

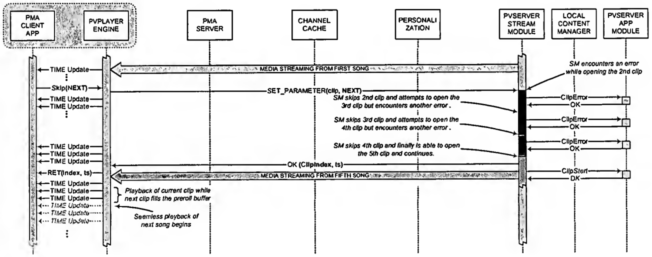
This section provides a walkthrough of the behavior when the SM encounters a problem with the next clip in the playlist when requested to skip to a new clip. The SM will skip over the problem clip to the next clip and try to continue streaming. The following call flow includes the streaming events sent to the AM to clarify the interaction of the event behavior along with the client RTSP behavior. Notice the addition of the clip index in the

SET_PARAMETER call from the SM to the client. This signals that the normal next clip was skipped and streaming is continuing with the third clip in the playlist.


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
TIME Update
Skip(NEXT)
TIME Update
TIME Update
RET(index, ts)
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
MEDIA STREAMING FROM FIRST SONG
SM encounters an error while opening the next clip
SET_PARAMETER(clip, NEXT)
ClipError
OK
SM skips 2nd clip and opens up the 3rd clip to begin streaming.
OK (ClipIndex, ts)
MEDIA STREAMING FROM THIRD SONG
ClipStart
OK
Playback of current clip while next clip fills the preroll buffer
Seemless playback of next song begins

A.2.4 Multiple Error In Skip Request

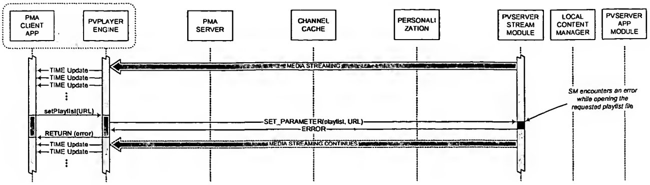
This section provides a walkthrough of the behavior when the SM encounters a problem with the multiple clips in the playlist. The SM will skip over the problem clip to the next clip and try to continue streaming. If successive problems are encountered, the SM will keep skipping to the next clip until either a good clip is found, the end of the playlist is reached, or a request to switch to a new playlist is received. The following call flow includes the streaming events sent to the AM to clarify the interaction of the event behavior along with the client RTSP behavior. Note that the SM will send a streaming event to the AM for every attempted clip.


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
TIME Update
Skip(NEXT)
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
RET(index, ts)
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
TIME Update
MEDIA STREAMING FROM FIRST SONG
SM encounters an error while opening the 2nd clip
SET_PARAMETER(clip, NEXT)
SM skips 2nd clip and attempts to open the 3rd clip but encounters another error.
SM skips 3rd clip and attempts to open the 4th clip but encounters another error.
SM skips 4th clip and finally is able to open the 5th clip and continues.
ClipError
OK
ClipError
OK
ClipError
OK
OK (ClipIndex, ts)
MEDIA STREAMING FROM FIFTH SONG
ClipStart
OK
Playback of current clip while next clip fills the preroll buffer
Seemless playback of next song begins

A.2.5 Error in Playlist Switch Request

This section provides a walkthrough of the behavior when the SM encounters a problem with the playlist file itself when requested to switch to a new playlist. The SM will simply return an error and continue streaming from the current clip from the current playlist until some other event occurs (i.e. a new playlist switch request, the playlist ends, etc.)

111


PMA CLIENT APP
PVPLAYER ENGINE
PMA SERVER
CHANNEL CACHE
PERSONALIZATION
PVSERVER STREAM MODULE
LOCAL CONTENT MANAGER
PVSERVER APP MODULE
MEDIA STREAMING
TIME Update
TIME Update
TIME Update
SM encounters an error while opening the requested playlist file
setPlaylist(URL)
SET_PARAMETER(playlist, URL)
ERROR
RETURN (error)
MEDIA STREAMING CONTINUES
TIME Update
TIME Update

112